```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>

#include <shastra/shastra.h>

#include <shastra/utils/list.h>

#include <shastra/datacomm/shastraDataH.h>
#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/datacomm/audioBiteH.h>
#include <shastra/datacomm/videoImgH.h>
#include <shastra/datacomm/ipimage.h>
#include <shastra/datacomm/xsCntlDataH.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>
#include <shastra/network/sharedMem.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>
#include <shastra/shautils/sesMgrFrontsP.h>
#include <shastra/shautils/clientHosts.h>

#include <shastra/front/frontP.h>
#include <shastra/front/front.h>
#include <shastra/front/frontState.h>
```

```
#include <shastra/front/frontCollClient.h>
#include <shastra/front/frontCollClientP.h>
#include <shastra/front/collabCntl.h>

#define USESHAREDMEM

#define checkConn() \
    if (pHostColl->fStatus == shaError) { \
        fprintf(stderr,"Connection to SesMgr is bad!\n"); \
        return -1; \
    }

#define sendReqString(s, arg) \
    if(hostSendQueuedRequest(pHostColl, s, arg) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr,"Error in Sending Shastra Operation Request\n"); \
        return -1; \
    }

#define sendDataString(s) \
    if(cmSendString(pHostColl->fdSocket, s) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr,"Error in Sending Shastra Operation Data\n"); \
        return -1; \
    }

#define ShastraIdIn(filedesc, pShaId) \
    if(shastraIdIn(pHostColl->fdSocket, pShaId) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving SID from SesMgr\n"); \
        return -1; \
    }

#define ShastraIdOut(filedesc, pShaId) \
    if(shastraIdOut(pHostColl->fdSocket, pShaId) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending SID to SesMgr\n"); \
        return -1; \
    }

#define ShastraIdsIn(filedesc, pShaIds) \
    if(shastraIdsIn(pHostColl->fdSocket, pShaIds) == -1){ \
        pHostColl->fStatus = shaError;  \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving SIDs from SesMgr\n"); \
        return -1; \
    }

#define ShastraIdsOut(filedesc, pShaIds) \
        if(shastraIdsOut(pHostColl->fdSocket, pShaIds) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending SIDs to SesMgr\n"); \
            return -1; \
        }

    #define ShastraIdTagIn(filedesc, pShaIdTag) \
        if(shastraIdTagIn(pHostColl->fdSocket, pShaIdTag) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving SIDTag from SesMgr\n"); \
            return -1; \
        }

    #define ShastraIdTagOut(filedesc, pShaIdTag) \
        if(shastraIdTagOut(pHostColl->fdSocket, pShaIdTag) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending SIDTag to SesMgr\n"); \
            return -1; \
        }

    #define ShastraIdTagsIn(filedesc, pShaIdTags) \
        if(shastraIdTagsIn(pHostColl->fdSocket, pShaIdTags) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving SIDTags from SesMgr\n"); \
            return -1; \
        }

    #define ShastraIdTagsOut(filedesc, pShaIdTags) \
        if(shastraIdTagsOut(pHostColl->fdSocket, pShaIdTags) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending SIDTags to SesMgr\n"); \
            return -1; \
        }

    #define ShastraULongOut(filedesc, pULong) \
        if(shaULongOut(pHostColl->fdSocket, pULong) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending pULong to SesMgr\n"); \
            return -1; \
        }

    #define ShastraULongIn(filedesc, pULong) \
        if(shaULongIn(pHostColl->fdSocket, pULong) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving pULong from SesMgr\n"); \
            return -1; \
    }

#define ShastraIntOut(filedesc, pInt) \
    if(shaIntOut(pHostColl->fdSocket, pInt) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending pInt to SesMgr\n"); \
        return -1; \
    }

#define ShastraIntIn(filedesc, pInt) \
    if(shaIntIn(pHostColl->fdSocket, pInt) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving pInt from SesMgr\n"); \
        return -1; \
    }


#define AudioBiteIn(filedesc, pABite) \
    if(audioBiteIn(pHostColl->fdSocket, pABite) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving ABite from SesMgr\n"); \
        return -1; \
    }

#define AudioBiteOut(filedesc, pABite) \
    if(audioBiteOut(pHostColl->fdSocket, pABite) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending ABite to SesMgr\n"); \
        return -1; \
    }


#define VideoImgIn(filedesc, pVImg) \
    if(videoImgIn(pHostColl->fdSocket, pVImg) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving VImg from SesMgr\n"); \
        return -1; \
    }

#define VideoImgOut(filedesc, pVImg) \
    if(videoImgOut(pHostColl->fdSocket, pVImg) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending VImg to SesMgr\n"); \
        return -1; \
    }
    #define TextBiteIn(filedesc, pTBite) \
        if(shaStringIn(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving TBite from SesMgr\n"); \
            return -1; \
        }

    #define TextBiteOut(filedesc, pTBite) \
        if(shaStringOut(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending TBite to SesMgr\n"); \
            return -1; \
        }

    #define PntrBiteIn(filedesc, pTBite) \
        if(shaDoublesIn(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving PntrB from SesMgr\n"); \
            return -1; \
        }

    #define PntrBiteOut(filedesc, pTBite) \
        if(shaDoublesOut(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending PntrB to SesMgr\n"); \
            return -1; \
        }

    #define CursorBiteIn(filedesc, pTBite) \
        if(shaDoublesIn(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving CursorB from SesMgr\n"); \
            return -1; \
        }

    #define CursorBiteOut(filedesc, pTBite) \
        if(shaDoublesOut(pHostColl->fdSocket, pTBite) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Sending CursorB to SesMgr\n"); \
            return -1; \
        }

    #define ImageDataIn(filedesc, pImage) \
        if(ipimageDataIn(pHostColl->fdSocket, pImage) == -1){ \
            pHostColl->fStatus = shaError; \
            closedChannelCleanupHandler(pHostColl->fdSocket); \
            fprintf(stderr, "Error Receiving image from SesMgr\n"); \
        return -1; \
    }

#define ImageDataOut(filedesc, pImage) \
    if(ipimageDataOut(pHostColl->fdSocket, pImage) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending image to SesMgr\n"); \
        return -1; \
    }


#define PictDataBitesIn(filedesc, pPCData) \
    if(pictPiecesIn(pHostColl->fdSocket, pPCData) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving PCData from SesMgr\n"); \
        return -1; \
    }

#define PictDataBitesOut(filedesc, pPCData) \
    if(pictPiecesOut(pHostColl->fdSocket, pPCData) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending PCData to SesMgr\n"); \
        return -1; \
    }

#define XSCntlBitesIn(filedesc, pXSCData) \
     if(xsCntlDatasIn(pHostColl->fdSocket, pXSCData) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Receiving PCData from SesMgr\n"); \
        return -1; \
    }

#define XSCntlBitesOut(filedesc, pXSCData) \
    if(xsCntlDatasOut(pHostColl->fdSocket, pXSCData) == -1){ \
        pHostColl->fStatus = shaError; \
        closedChannelCleanupHandler(pHostColl->fdSocket); \
        fprintf(stderr, "Error Sending PCData to SesMgr\n"); \
        return -1; \
    }


cmCommand       frontCollCmdTab[] = FRONTCOLL_CLIENTCMDS;
#define NFRONTCOLL_CLIENTCMDS (sizeof(frontCollCmdTab)/sizeof(cmCommand))
/* number of commands */
int             frontCollNCmds = NFRONTCOLL_CLIENTCMDS;

cmCommand       frontCollInCmdTab[] = FRONTCOLL_CLIENTINCMDS;
#define NFRONTCOLL_CLIENTINCMDS (sizeof(frontCollInCmdTab)/sizeof(cmCommand
    ))
```

```
/* number of commands */
int             frontCollNInCmds = NFRONTCOLL_CLIENTINCMDS;

shaCmdData      frontCollCmdData;


/*
 * Function
 */
int
collTellLeaderRespHandler(fd)
     int             fd;
{
  shastraIdTag    sIdTag, sesmSIdTag;
  unsigned long   lIdTag;
  hostData *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collTellLeaderRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &sesmSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  ShastraULongIn(fd, &lIdTag);

  collabSetLeaderOprn(sIdTag, sesmSIdTag, lIdTag);

  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_TELLLEADER);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collTerminateReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_COLL_TERMINATE, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}


/*
 * Function
 */
int
collTerminateRespHandler(fd)
```

```
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (collabTerminateFunc != NULL) {
    (*collabTerminateFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"collabTerminateFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_TERMINATE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collHelpReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_HELP, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collHelpRespHandler(fd)
     int             fd;
{
  standardHelpRespHandler(fd);

  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_HELP);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collJoinReq(pHost, pSId, pPermTag, pCmdData)
     hostData       *pHost;
     shastraId      *pSId;
     shastraIdTag   *pPermTag;
     shaCmdData *pCmdData;
{
    int              collSocket;
    int              status;
    hostData        *pHostColl;

    if((pSId == NULL) || (pPermTag == NULL)){
      fprintf(stderr, "collJoinReq()->bad args!\n");
      return -1;
    }
    if(pCmdData == NULL){
      fprintf(stderr, "collJoinReq()->Warning: No Control Data!\n");
    }
    status = cmClientConnect2Server(pSId->nmHost, pSId->nmApplicn,
                     pSId->iPort, &collSocket);
    if (status == -1) {
      sprintf(pFrontAppData->sbMsgBuf, "collJoinReq()-- Couldn't connect\n");
      showCollabInfo(pFrontAppData->sbMsgBuf);
      return -1;
    } else {
      sprintf(pFrontAppData->sbMsgBuf, "collJoinReq()-- connected\n");
      showCollabInfo(pFrontAppData->sbMsgBuf);
    }

    pHostColl = (hostData *) malloc(sizeof(hostData));
    memset(pHostColl, 0, sizeof(hostData));
    pHostColl->fdSocket = collSocket;
    pHostColl->lSIDTag = pSId->lSIDTag;
    pHostColl->pSId = copyId(pSId, NULL);
    pHostColl->sendList = listMakeNew();
    pHostColl->recvList = listMakeNew();
    pHostColl->fStatus = shaWait2Send;

    if (frontCollCmdData.pCmdTab == NULL) {
      memset(&frontCollCmdData, 0, sizeof(shaCmdData));
      frontCollCmdData.pCmdTab = frontCollCmdTab;
      frontCollCmdData.nCmds = frontCollNCmds;
      frontCollCmdData.pCmdTabIn = frontCollInCmdTab;
      frontCollCmdData.nCmdsIn = frontCollNInCmds;
  /*CHECK, will allow only one kind of collab*/
  /*add cmd data once per session type*/
      cmJoinCmdData(&frontCollCmdData, pCmdData);
    }

    shaKernFlags[collSocket] = SHASESMGR;

    if (mplexRegisterChannel(pHostColl->fdSocket, shaClientHandler,
                  &frontCollCmdData, (char *) pHostColl) == -1) {
      fprintf(stderr, "collJoinReq()->Couldn't Register Client Handler!!\n");
      pHostColl->fStatus = shaError;
      return -1;
    }
    mplexSetHostData(pHostColl->fdSocket, pHostColl);
    if((pHost = mplexGetHostData(pHostColl->fdSocket)) != pHostColl){
      fprintf(stderr,"collJoinReq()->mplexSetHostData problem!\n");
  }

  pFrontSId->lPerms = *pPermTag;

  checkConn();
  sendReqString(REQ_COLL_JOIN, NULL);
  ShastraIdOut(pHostColl->fdSocket, pFrontSId);
  cmFlush(pHostColl->fdSocket);

  collabSetCurrHostOprn(pHostColl, False);
  /* if no current, created becomes current */
  return 0;
}

/*
 * Function
 */
int
collJoinRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraId      *pSId;
  int             iLocCollabSelect;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collJoinRespHandler()->NULL Host data!\n");
    return -1;
  }
  pSId = getSIdByTagInSIds(&pHostColl->lSIDTag, &shastraSesmIds);
  if (pSId == NULL) {
    fprintf(stderr, "collJoinRespHandler()->Missing SesMgr! Aborting\n");
    return -1;
  }
  if ((iLocCollabSelect = locateClientHosts(pSId)) == -1) {
    iLocCollabSelect = occupyClHostFreeSlot(pSId);
  }
  updateAddClHost(pSId, pHostColl);
  collabSetCurrHostOprn(pHostColl, False);
  /* if no current, created becomes current */

  setCollabNamesOprn(pSId->lSIDTag);

  if (collabJoinFunc != NULL) {
    (*collabJoinFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"collabJoinFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_JOIN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
```

```
}

/*
 * Function
 */
collAskJnRespHandler(fd)
     int             fd;
{
  shastraIdTag    sesmSIdTag;
  shastraIdTag    frontSIdTag;
  hostData *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collAskJnRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &sesmSIdTag);
  ShastraIdTagIn(fd, &frontSIdTag);

  /* now prompt for participation, and tell join if reqd */
/*
  collAskJoinPromptOprn(sesmSIdTag, frontSIdTag);
  */
  collabAskJoinPromptOprn(sesmSIdTag, frontSIdTag);

  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJOIN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collAskJnMsgRespHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;
  hostData *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collAskJnMsgRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabRecvdAskJoinMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJOINMSG
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int
collLeaveReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_COLL_LEAVE, NULL);
  cmFlush(pHostColl->fdSocket);
  collLeaveRespHandler(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collLeaveRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraId      *pSIdHost;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collLeaveRespHandler()->NULL Host data!\n");
    return -1;
  }

  pSIdHost = getSIdByTagInSIds(&pHostColl->lSIDTag, &shastraSesmIds);
  if (pSIdHost == NULL) {
    fprintf(stderr, "collLeaveRespHandler()->Missing SesMgr! Aborting\n");
    return -1;
  }
  updateRmvClHostByIdTag(pSIdHost, &pHostColl->lSIDTag);

  setCollabNamesOprn(pHostColl->lSIDTag);

  /* close connection */
  mplexUnRegisterChannel(fd);
  if (collabLeaveFunc != NULL) {
    (*collabLeaveFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"collabLeaveFunc()->no handler!\n");
  }
  collabResetCurrHostOprn(pHostColl, False);

#ifdef CLEANLYREMOVE
  listDestroy(pHostColl->sendList, 1);
  listDestroy(pHostColl->recvList, 1);
  memset(pHostColl, 0, sizeof(hostData));
  /*is freed in shaClientHandler ! ugh!!*/
#endif /* CLEANLYREMOVE */
    if(pHostColl->pSId != NULL){
      shastraIdXDRFree(pHostColl->pSId);
    }
  free(pHostColl);

  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_LEAVE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRemoveReq(pHostColl, pSIdTag)
     hostData       *pHostColl;
     shastraIdTag   *pSIdTag;
{
  checkConn();
  sendReqString(REQ_COLL_REMOVE, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  cmFlush(pHostColl->fdSocket);
  return 0;
}


/*
 * Function
 */
int
collRemoveRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (collabRemoveFunc != NULL) {
    (*collabRemoveFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"collabRemoveFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_REMOVE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSetPermsReq(pHostColl, pSIdTag, perms)
     hostData        *pHostColl;
     shastraIdTag    *pSIdTag;
     unsigned long    perms;
{
  checkConn();
  sendReqString(REQ_SET_COLLPERMS, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  ShastraULongOut(pHostColl->fdSocket, &perms);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSetPermsRespHandler(fd)
     int              fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    sIdTag;
  shastraIdTag    permTag;
  shastraIdTags  *pPermTags, *pFrIdTags;
  hostData       *pHostColl;
  char           *tmp;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSetPermsRespHandler()->NULL Host data!\n");
    return -1;
  }

  smSIdTag = pHostColl->lSIDTag;
  ShastraIdTagIn(fd, &sIdTag);
  ShastraIdTagIn(fd, &permTag);
  pFrIdTags = getSesmFrontSIdTags(&smSIdTag);
  pPermTags = getSesmFrontPermTags(&smSIdTag);

  if (setSesmFrontPerms(&smSIdTag, &sIdTag, permTag) < 0) {
    fprintf(stderr, "collSetPermsRespHandler()->can't set perms for %lx!\n"
        ,
        sIdTag);
  }
  if(sIdTag == pFrontSId->lSIDTag){
    setCollabFrontPermsOprn(smSIdTag);
  }
  if (collabSetPermsFunc != NULL) {
    (*collabSetPermsFunc) (pHostColl, &sIdTag, permTag);
  }
  else{
    fprintf(stderr,"collabSetPermsFunc()->no handler!\n");
  }
  tmp = perms2Str(permTag);
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s(%s)\n", tmp,
      REQ_SET_COLLPERMS);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(tmp);
  return 0;
}

/*
 * Function
 */
int
collGetPermsReq(pHostColl, pSIdTag)
     hostData       *pHostColl;
     shastraIdTag   *pSIdTag;
{
  checkConn();
  sendReqString(REQ_GET_COLLPERMS, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collGetPermsRespHandler(fd)
     int             fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    sIdTag;
  shastraIdTag    permTag;
  shastraIdTags  *pPermTags, *pFrIdTags;
  hostData       *pHostColl;
  char           *tmp;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collGetPermsRespHandler()->NULL Host data!\n");
    return -1;
  }

  smSIdTag = pHostColl->lSIDTag;
  ShastraIdTagIn(fd, &sIdTag);
  ShastraIdTagIn(fd, &permTag);
  pFrIdTags = getSesmFrontSIdTags(&smSIdTag);
  pPermTags = getSesmFrontPermTags(&smSIdTag);
  if (setSesmFrontPerms(&smSIdTag, &sIdTag, permTag) < 0) {
    fprintf(stderr, "collGetPermsRespHandler()->can't set perms for %lx!\n"
        ,
        sIdTag);
  }
  if(sIdTag == pFrontSId->lSIDTag){
    setCollabFrontPermsOprn(smSIdTag);
  }
  if (collabGetPermsFunc != NULL) {
    (*collabGetPermsFunc) (pHostColl, &sIdTag, permTag);
  }
  else{
    fprintf(stderr,"collabGetPermsFunc()->no handler!\n");
  }
  tmp = perms2Str(permTag);
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s(%s)\n", tmp,
      REQ_GET_COLLPERMS);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(tmp);
  return 0;
}

/*
 * Function
 */
int
collGetSesmPermsReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_GET_COLLPERMS, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collGetSesmPermsRespHandler(fd)
     int             fd;
{
  shastraIdTag    smSIdTag;
  static shastraIdTags permTags;
  shastraIdTags  *pPermTags;
  int             smIndex;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collGetSesmPermsRespHandler()->NULL Host data!\n");
```

```
      return -1;
    }

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagsIn(fd, &permTags);
    smIndex = locateSesmFronts(&smSIdTag);
    if (smIndex == -1) {
      fprintf(stderr, "collGetSesmPermsRespHandler()->can't locate sesMgr!\n"
          );
    } else {
      pPermTags = getSesmFrontPermTags(&smSIdTag);
      if (pPermTags->shastraIdTags_len == permTags.shastraIdTags_len) {
        shastraIdTag   *pSIdTag;

        /* just switch, should be ok */
        pSIdTag = pPermTags->shastraIdTags_val;
        pPermTags->shastraIdTags_val = permTags.shastraIdTags_val;
        permTags.shastraIdTags_val = pSIdTag;
      }
    }
    sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SESMCOLLPERMS);
    showCollabInfo(pFrontAppData->sbMsgBuf);

    return 0;
  }


  /*
   * Function
   */
  int
  collSetSesmPermsReq(pHostColl, pPermTags)
       hostData       *pHostColl;
       shastraIdTags  *pPermTags;
  {
    checkConn();
    sendReqString(REQ_SET_SESMCOLLPERMS, NULL);
    ShastraIdTagsOut(pHostColl->fdSocket, pPermTags);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collSetSesmPermsRespHandler(fd)
       int             fd;
  {
    sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SET_SESMCOLLPERMS);
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }
```

```
/*
 * Function
 */
int
collGetIxnModeReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_GET_IXNMODE, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;

}

/*
 * Function
 */
int
collGetIxnModeRespHandler(fd)
     int              fd;
{
  unsigned long   ixnMode;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collGetIxnModeRespHandler()->NULL Host data!\n");
    return -1;
  }
  ShastraULongIn(fd, &ixnMode);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->lIxnMode = ixnMode;
  if (collabGetIxnModeFunc != NULL) {
    (*collabGetIxnModeFunc) (pHostColl, ixnMode);
  }
  else{
    fprintf(stderr,"collabGetIxnModeFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_IXNMODE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSetIxnModeReq(pHostColl, ixnMode)
     hostData       *pHostColl;
     unsigned long   ixnMode;
{
  checkConn();
  sendReqString(REQ_SET_IXNMODE, NULL);
  ShastraULongOut(pHostColl->fdSocket, &ixnMode);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSetIxnModeRespHandler(fd)
     int              fd;
{
  unsigned long   ixnMode;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSetIxnModeRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraULongIn(fd, &ixnMode);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->lIxnMode = ixnMode;

  if (collabSetIxnModeFunc != NULL) {
    (*collabSetIxnModeFunc) (pHostColl, ixnMode);
  }
  else{
    fprintf(stderr,"collabSetIxnModeFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SET_IXNMODE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collGetFloorModeReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_GET_FLOORMODE, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;

}
```

```
/*
 * Function
 */
int
collGetFloorModeRespHandler(fd)
     int             fd;
{
  unsigned long   floorMode;
  hostData      *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collGetFloorModeRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraULongIn(fd, &floorMode);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->lFloorMode = floorMode;

  if (collabGetFloorModeFunc != NULL) {
    (*collabGetFloorModeFunc) (pHostColl, floorMode);
  }
  else{
    fprintf(stderr,"collabGetFloorModeFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_FLOORMODE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSetFloorModeReq(pHostColl, ixnMode)
     hostData       *pHostColl;
     unsigned long   ixnMode;
{
  checkConn();
  sendReqString(REQ_SET_FLOORMODE, NULL);
  ShastraULongOut(pHostColl->fdSocket, &ixnMode);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSetFloorModeRespHandler(fd)
     int             fd;
```

```
{
  unsigned long   floorMode;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSetFloorModeRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraULongIn(fd, &floorMode);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->lFloorMode = floorMode;

  if (collabSetFloorModeFunc != NULL) {
    (*collabSetFloorModeFunc) (pHostColl, floorMode);
  }
  else{
    fprintf(stderr,"collabSetFloorModeFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SET_FLOORMODE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}



/*
 * Function
 */
int
collGetSesFormatReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_GET_SESFORMAT, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;

}

/*
 * Function
 */
int
collGetSesFormatRespHandler(fd)
     int             fd;
{
  unsigned long   sesFormat;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;
  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collGetSesFormatRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraULongIn(fd, &sesFormat);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->lFormat = sesFormat;

  if (collabGetFormatFunc != NULL) {
    (*collabGetFormatFunc) (pHostColl, sesFormat);
  }
  else{
    fprintf(stderr,"collabGetFormatFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SESFORMAT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSetSesFormatReq(pHostColl, sesFormat)
     hostData       *pHostColl;
     unsigned long   sesFormat;
{
  checkConn();
  sendReqString(REQ_SET_SESFORMAT, NULL);
  ShastraULongOut(pHostColl->fdSocket, &sesFormat);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSetSesFormatRespHandler(fd)
     int             fd;
{
  unsigned long   sesFormat;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSetSesFormatRespHandler()->NULL Host data!\n");
    return -1;
  }
```

```
    ShastraULongIn(fd, &sesFormat);
    pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
    pSesmFrCD->lFormat = sesFormat;

    if (collabSetFormatFunc != NULL) {
      (*collabSetFormatFunc) (pHostColl, sesFormat);
    }
    else{
      fprintf(stderr,"collabSetFormatFunc()->no handler!\n");
    }
    sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SET_SESFORMAT);
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }


  /*
   * Function
   */
  int
  collGrabTokenReq(pHostColl)
       hostData       *pHostColl;
  {
    checkConn();
    sendReqString(REQ_GRAB_TOKEN, NULL);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collGrabTokenRespHandler(fd)
       int              fd;
  {
    hostData       *pHostColl;
    shastraIdTag    sIdTagToken;
    sesmFronts *pSesmFrCD;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collGrabTokenRespHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &sIdTagToken);
    pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
    pSesmFrCD->sIdTagToken = sIdTagToken;

    setCollabFrontFloorOprn(pHostColl->lSIDTag, sIdTagToken);
    if (collabGrabTokenFunc != NULL) {
      (*collabGrabTokenFunc) (pHostColl, &sIdTagToken);
  }
  else{
    fprintf(stderr,"collabGrabTokenFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GRAB_TOKEN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collFreeTokenReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_FREE_TOKEN, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}



/*
 * Function
 */
int
collFreeTokenRespHandler(fd)
     int             fd;
{
  shastraIdTag    sIdTagToken = 0;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collFreeTokenRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (collabFreeTokenFunc != NULL) {
    (*collabFreeTokenFunc) (pHostColl, &sIdTagToken);
  }
  else{
    fprintf(stderr,"collabFreeTokenFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_FREE_TOKEN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
```

```
int
collTellTokenReq(pHostColl, pSIdTag)
     hostData       *pHostColl;
     shastraIdTag *pSIdTag;
{
  checkConn();
  sendReqString(REQ_TELL_TOKEN, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collTellTokenRespHandler(fd)
     int             fd;
{
  shastraIdTag    sIdTagToken = 0;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collTellTokenRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (collabTellTokenFunc != NULL) {
    (*collabTellTokenFunc) (pHostColl, &sIdTagToken);
  }
  else{
    fprintf(stderr,"collabTellTokenFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_TELL_TOKEN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collAskTokenReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_ASK_TOKEN, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}
```

```
/*
 * Function
 */
int
collAskTokenRespHandler(fd)
     int              fd;
{
  shastraIdTag    sIdTagToken;
  hostData       *pHostColl;
  sesmFronts *pSesmFrCD;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collAskTokenRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &sIdTagToken);
  pSesmFrCD = getSesMgrCntlData(&pHostColl->lSIDTag);
  pSesmFrCD->sIdTagToken = sIdTagToken;

  setCollabFrontFloorOprn(pHostColl->lSIDTag, sIdTagToken);
  if (collabAskTokenFunc != NULL) {
    (*collabAskTokenFunc) (pHostColl, &sIdTagToken);
  }
  else{
    fprintf(stderr,"collabAskTokenFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_ASK_TOKEN);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}



/*
 * Function
 */
int
collStartTextReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_TEXT, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartTextRespHandler(fd)
```

```
     int             fd;
{
  /* start a text comm infrastructure.. one text wid per member */
  /* create and popup text comm controller */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (textStartFunc != NULL) {
    (*textStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"textStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_TEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndTextReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_TEXT, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndTextRespHandler(fd)
     int             fd;
{
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;
  /* terminate a text comm channel destroy wids etc */
  /* destroy popdown text comm controller */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (textEndFunc != NULL) {
    (*textEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"textEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_TEXT);
```

```
      showCollabInfo(pFrontAppData->sbMsgBuf);
      return 0;
    }

    /*
     * Function
     */
    int
    collSendTextReq(pHostColl, nameBuf)
         hostData       *pHostColl;
         char           *nameBuf;
    {
      checkConn();
      sendReqString(REQ_SEND_TEXT, NULL);
      sendDataString(nameBuf);
      cmFlush(pHostColl->fdSocket);
      return 0;
    }

    /*
     * Function
     */
    int
    collSendTextRespHandler(fd)
         int             fd;
    {
      hostData       *pHostColl;

      pHostColl = mplexGetHostData(fd);
      if (pHostColl == NULL) {
        fprintf(stderr, "collSendTextRespHandler()->NULL Host data!\n");
        return -1;
      }

      if (textSendFileFunc != NULL) {
        (*textSendFileFunc) (pHostColl);
      }
      else{
        fprintf(stderr,"textSendFileFunc()->no handler!\n");
      }
      sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_TEXT);
      showCollabInfo(pFrontAppData->sbMsgBuf);
      return 0;
    }

    /*
     * Function
     */
    int
    collSendTextInHandler(fd)
         int             fd;
    {
      /* recv msg from outside.. update local view */
```

```
  hostData       *pHostColl;
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_TEXT);
  if (textRecvFileFunc != NULL) {
    (*textRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"textRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgTextReq(pHostColl, str)
     hostData       *pHostColl;
     char           *str;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORTEXT
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = strlen(str) + 1;
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgTextReq()->couldn't shMemReuseSegment!\n"
          );
    }
    memcpy(pShmInfo->shmAddr, str, n);
    collSendMsgShmTextReq(pHostColl, pShmInfo);
    return -1;
  }
#endif              /* USESHAREDMEMFORTEXT */

  checkConn();
  sendReqString(REQ_SEND_MSGTEXT, NULL);
  sendDataString(str);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgTextRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgTextRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (textSendMsgFunc != NULL) {
    (*textSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"textSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgTextInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  if (textRecvMsgFunc != NULL) {
    (*textRecvMsgFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"textRecvMsgFunc()->no handler!\n");
  }
  free(buf);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgTextReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_RECVD_MSGTEXT, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgTextRespHandler(fd)
     int             fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgTextInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmTextReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmTextReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMTEXT, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmTextRespHandler(fd)
     int            fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmTextRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (textSendMsgFunc != NULL) {
    (*textSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"textSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmTextInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmTextInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
    fprintf(stderr, "collSendMsgShmTextInHandler()->SHM recon problem\n");
    return -1;
  }
  buf = pShmInfo->shmAddr;
  if (textRecvMsgFunc != NULL) {
    (*textRecvMsgFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"textRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGSHMTEXT
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
```

```
int
collRecvdMsgShmTextReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmTextReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMTEXT, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmTextRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmTextInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmTextInHandler()->couldn't shMemDelete!\
        n");
    }
    /* recvd ack that all collabs have heard, delete shared seg */
    sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGSHMTEXT
        );
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }

  /*
   * Function
   */
  int
  collStartAudioReq(pHostColl)
       hostData       *pHostColl;
  {
    checkConn();
    sendReqString(REQ_START_AUDIO, NULL);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collStartAudioRespHandler(fd)
       int             fd;
  {
    /* start a audio comm infrastructure.. */
    shastraIdTag    senderSIdTag;
    hostData       *pHostColl;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collStartAudioRespHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);
    if (audioStartFunc != NULL) {
      (*audioStartFunc) (pHostColl, &senderSIdTag);
    }
    else{
      fprintf(stderr,"audioStartFunc()->no handler!\n");
    }
    sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_AUDIO);
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }

  /*
   * Function
   */
```

```
int
collEndAudioReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_AUDIO, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndAudioRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  /* terminate a audio comm channel */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (audioEndFunc != NULL) {
    (*audioEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"audioEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_AUDIO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendAudioReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_AUDIO, NULL);
  sendDataString(nameBuf);

  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
```

```
collSendAudioRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendAudioRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (audioSendFileFunc != NULL) {
    (*audioSendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"audioSendFileFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_AUDIO);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendAudioInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendAudioInHandler()->NULL Host data!\n");
    return -1;
  }

  /*ShastraIdTagIn(fd, &senderSIdTag);*/

  buf = cmReceiveString(fd);
  if (audioRecvFileFunc != NULL) {
    (*audioRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"audioRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_AUDIO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}
/*
 * Function
 */
int
collSendMsgAudioReq(pHostColl, pABite)
     hostData       *pHostColl;
     audioBite      *pABite;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORAUDIO
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = pABite->data.data_len + sizeof(audioBite);
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgAudioReq()->couldn't shMemReuseSegment!\
          n");
    }
    /* xdr dump */
    audioBiteMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pABite);
    collSendMsgShmAudioReq(pHostColl, pShmInfo);
    return -1;
  }
#endif              /* USESHAREDMEMFORAUDIO */
  checkConn();
  sendReqString(REQ_SEND_MSGAUDIO, NULL);
  AudioBiteOut(pHostColl->fdSocket, pABite);
  /*
   * nameBuf = (char*)pABite; sendDataString(nameBuf);
   */
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgAudioRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgAudioRespHandler()->NULL Host data!\n");
    return -1;
    }

    if (audioSendMsgFunc != NULL) {
      (*audioSendMsgFunc) (pHostColl);
    }
    else{
      fprintf(stderr,"audioSendMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGAUDIO);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collSendMsgAudioInHandler(fd)
       int             fd;
  {
    hostData       *pHostColl;
    /* recv msg from outside.. update local view */
    char           *buf;
    shastraIdTag    senderSIdTag;
    static audioBite aBite;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendMsgAudioInHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);

    memset(&aBite, 0, sizeof(audioBite));
    AudioBiteIn(fd, &aBite);
    if (audioRecvMsgFunc != NULL) {
      (*audioRecvMsgFunc) (pHostColl, &senderSIdTag, &aBite);
    }
    else{
      fprintf(stderr,"audioRecvMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
         REQ_SEND_MSGAUDIO);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
```

```
 * Function
 */
int
collRecvdMsgAudioReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_RECVD_MSGAUDIO, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgAudioRespHandler(fd)
     int             fd;
{
  /* NULL -- recvd msg got to sesmgr */
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGAUDIO);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}
/*
 * Function
 */
int
collRecvdMsgAudioInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgAudioInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGAUDIO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmAudioReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmAudioReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMAUDIO, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmAudioRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmAudioRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (audioSendMsgFunc != NULL) {
    (*audioSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"audioSendMsgFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMAUDIO)
       ;
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendMsgShmAudioInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;
  static audioBite aBite;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmAudioInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmAudioInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
    fprintf(stderr, "collSendMsgShmAudioInHandler()->SHM recon problem\n");
    return -1;
  }
  audioBiteMemIn(pShmInfo->shmAddr, pShmInfo->shmSize, &aBite);
  if (audioRecvMsgFunc != NULL) {
    (*audioRecvMsgFunc) (pHostColl, &senderSIdTag, &aBite);
  }
  else{
    fprintf(stderr,"audioRecvMsgFunc()->no handler!\n");
  }
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
       REQ_SEND_MSGSHMAUDIO);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmAudioReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmAudioReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMAUDIO, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmAudioRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMAUDIO
      );
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collRecvdMsgShmAudioInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  shmInfo        *pShmInfo;
  int             shmId;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmAudioInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmAudioInHandler()->couldn't shMemDelete!
        \n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
```

```
      sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n",
          REQ_RECVD_MSGSHMAUDIO);
      showCollabInfo(pFrontAppData->sbMsgBuf);
      return 0;
    }

    /*
     * Function
     */
    int
    collStartVideoReq(pHostColl)
         hostData       *pHostColl;
    {
      checkConn();
      sendReqString(REQ_START_VIDEO, NULL);
      cmFlush(pHostColl->fdSocket);
      return 0;
    }

    /*
     * Function
     */
    int
    collStartVideoRespHandler(fd)
         int              fd;
    {
      hostData       *pHostColl;
      shastraIdTag    senderSIdTag;
      /* start a video comm infrastructure.. start video controller etc */
      /* create and popup video comm controller */
      pHostColl = mplexGetHostData(fd);
      ShastraIdTagIn(fd, &senderSIdTag);
      if (videoStartFunc != NULL) {
        (*videoStartFunc) (pHostColl, &senderSIdTag);
      }
      else{
        fprintf(stderr,"videoStartFunc()->no handler!\n");
      }
      sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_VIDEO);
      showCollabInfo(pFrontAppData->sbMsgBuf);
      return 0;
    }

    /*
     * Function
     */
    int
    collEndVideoReq(pHostColl)
         hostData       *pHostColl;
    {
      checkConn();
      sendReqString(REQ_END_VIDEO, NULL);
      cmFlush(pHostColl->fdSocket);
```

```
  return 0;
}

/*
 * Function
 */
int
collEndVideoRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  /* terminate a video comm channel destroy controller */
  /* destroy popdown video comm controller */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (videoEndFunc != NULL) {
    (*videoEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"videoEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_VIDEO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendVideoReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_VIDEO, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendVideoRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
      fprintf(stderr, "collSendVideoRespHandler()->NULL Host data!\n");
      return -1;
    }

    if (videoSendFileFunc != NULL) {
      (*videoSendFileFunc) (pHostColl);
    }
    else{
      fprintf(stderr,"videoSendFileFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_VIDEO);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collSendVideoInHandler(fd)
       int              fd;
  {
    hostData       *pHostColl;
    char           *nameBuf;
    shastraIdTag    senderSIdTag;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendVideoInHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);
    /* recv msg from outside.. update local view */
    nameBuf = cmReceiveString(fd);
    if (videoRecvFileFunc != NULL) {
      (*videoRecvFileFunc) (pHostColl, &senderSIdTag, nameBuf);
    }
    else{
      fprintf(stderr,"videoRecvFileFunc()->no handler!\n");
    }
    sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_VIDEO);
    showCollabInfo(pFrontAppData->sbMsgBuf);

    free(nameBuf);
    return 0;
  }

  /*
   * Function
   */
```

```
int
collSendMsgVideoReq(pHostColl, pVImg)
     hostData       *pHostColl;
     videoImg       *pVImg;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEM
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = pVImg->data.data_len + sizeof(videoImg);
    if (shMemReuseSegment(pShmInfo, ((n > 65536) ? n : 65536)) == 0) {
      fprintf(stderr, "collSendMsgVideoReq()->couldn't shMemReuseSegment!\
          n");
    }
    /* xdr dump */
    videoImgMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pVImg);
    collSendMsgShmVideoReq(pHostColl, pShmInfo);
    return 0;
  }
#endif              /* USESHAREDMEM */
  checkConn();
  sendReqString(REQ_SEND_MSGVIDEO, NULL);
  VideoImgOut(pHostColl->fdSocket, pVImg);
  /*
   * nameBuf = (char*)pVImg; sendDataString(nameBuf);
   */
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgVideoRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgVideoRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (videoSendMsgFunc != NULL) {
    (*videoSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"videoSendMsgFunc()->no handler!\n");
  }
```

```
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGVIDEO);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collSendMsgVideoInHandler(fd)
       int              fd;
  {
    hostData       *pHostColl;
    shastraIdTag    senderSIdTag;
    char           *nameBuf;
    static videoImg vImg;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendMsgVideoInHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);
    /* recv msg from outside.. update local view */
    VideoImgIn(fd, &vImg);
    if (videoRecvMsgFunc != NULL) {
      (*videoRecvMsgFunc) (pHostColl, &senderSIdTag, &vImg);
    }
    else{
      fprintf(stderr,"videoRecvMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
         REQ_SEND_MSGVIDEO);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collRecvdMsgVideoReq(pHostColl, nameBuf)
       hostData       *pHostColl;
       char           *nameBuf;
  {
    checkConn();
    sendReqString(REQ_RECVD_MSGVIDEO, NULL);
    sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgVideoRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGVIDEO);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collRecvdMsgVideoInHandler(fd)
     int              fd;
{
  hostData        *pHostColl;
  char            *nameBuf;
  shastraIdTag     senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgVideoInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGVIDEO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmVideoReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmVideoReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMVIDEO, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmVideoRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmVideoRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (videoSendMsgFunc != NULL) {
    (*videoSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"videoSendMsgFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMVIDEO)
       ;
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendMsgShmVideoInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  shastraIdTag    senderSIdTag;
  int             shmId;
  static videoImg vImg;
  shmInfo       *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmVideoInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmVideoInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
    fprintf(stderr, "collSendMsgShmVideoInHandler()->SHM recon problem\n");
    return -1;
  }
  videoImgMemIn(pShmInfo->shmAddr, pShmInfo->shmSize, &vImg);
  if (videoRecvMsgFunc != NULL) {
    (*videoRecvMsgFunc) (pHostColl, &senderSIdTag, &vImg);
  }
  else{
    fprintf(stderr,"videoRecvMsgFunc()->no handler!\n");
  }
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_SEND_MSGSHMVIDEO);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmVideoReq(pHostColl, pShmInfo)
     hostData      *pHostColl;
     shmInfo       *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmVideoReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMVIDEO, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
```

```
}

/*
 * Function
 */
int
collRecvdMsgShmVideoRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMVIDEO
       );
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collRecvdMsgShmVideoInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmVideoInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmVideoInHandler()->couldn't shMemDelete!
        \n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n",
      REQ_RECVD_MSGSHMVIDEO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
```

```
int
collStartPolyReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_POLY, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartPolyRespHandler(fd)
     int             fd;
{
  /* start a image comm infrastructure.. one image wid per member */
  /* create and popup image comm controller */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (polyStartFunc != NULL) {
    (*polyStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"polyStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_POLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndPolyReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_POLY, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndPolyRespHandler(fd)
```

```
     int             fd;
{
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;
  /* terminate a image comm channel destroy wids etc */
  /* destroy popdown image comm controller */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (polyEndFunc != NULL) {
    (*polyEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"polyEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_POLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendPolyReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_POLY, NULL);
  sendDataString(nameBuf);

  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendPolyRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPolyRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (polySendFileFunc != NULL) {
    (*polySendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"polySendFileFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_POLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendPolyInHandler(fd)
     int              fd;
{
  /* recv msg from outside.. update local view */
  hostData       *pHostColl;
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPolyInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_POLY);
  if (polyRecvFileFunc != NULL) {
    (*polyRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"polyRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPolyReq(pHostColl, pImage)
     hostData       *pHostColl;
     ipimageData    *pImage;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORMPOLY
```

```
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = pImage->mPoly->nPolygons * 100 * sizeof(double);
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgPolyReq()->couldn't shMemReuseSegment!\n"
          );
    }
    /* xdr dump */
    ipimageDataMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pImage);
    collSendMsgShmPolyReq(pHostColl, pShmInfo);
    return 0;
  }
#endif              /* USESHAREDMEMFORMPOLY */
  checkConn();
  sendReqString(REQ_SEND_MSGPOLY, NULL);
  ImageDataOut(pHostColl->fdSocket, pImage);
  /*
   * nameBuf = (char*)pImage; sendDataString(nameBuf);
   */
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPolyRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPolyRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (polySendMsgFunc != NULL) {
    (*polySendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"polySendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGPOLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
```

```
collSendMsgPolyInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  ipimageData    *pImage;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPolyInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  pImage = (ipimageData *) malloc(sizeof(ipimageData));
  memset(pImage, 0, sizeof(ipimageData));
  ImageDataIn(fd, pImage);
  if (polyRecvMsgFunc != NULL) {
    (*polyRecvMsgFunc) (pHostColl, &senderSIdTag, pImage);
  }
  else{
    fprintf(stderr,"polyRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGPOLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPolyReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_RECVD_MSGPOLY, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPolyRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPOLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPolyInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgPolyInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGPOLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPolyReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmPolyReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMPOLY, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
```

```
 * Function
 */
int
collSendMsgShmPolyRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmPolyRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (polySendMsgFunc != NULL) {
    (*polySendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"polySendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPOLY);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPolyInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  ipimageData    *pImage;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmPolyInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmPolyInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
```

```
    if (!shMemReconnect(pShmInfo, shmId)) {
      fprintf(stderr, "collSendMsgShmPolyInHandler()->SHM recon problem\n");
      return -1;
    }
    pImage = (ipimageData *) malloc(sizeof(ipimageData));
    memset(pImage, 0, sizeof(ipimageData));
    ipimageDataMemIn(pShmInfo->shmAddr, pShmInfo->shmSize, pImage);
    if (polyRecvMsgFunc != NULL) {
      (*polyRecvMsgFunc) (pHostColl, &senderSIdTag, pImage);
    }
    else{
      fprintf(stderr,"polyRecvMsgFunc()->no handler!\n");
    }
    sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGSHMPOLY
        );
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }
  /*
   * Function
   */
  int
  collRecvdMsgShmPolyReq(pHostColl, pShmInfo)
       hostData       *pHostColl;
       shmInfo        *pShmInfo;
  {
    if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
      fprintf(stderr, "collRecvdMsgShmPolyReq()->no non-local SHM\n");
      return -1;
    }
    checkConn();
    sendReqString(REQ_RECVD_MSGSHMPOLY, NULL);
    ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collRecvdMsgShmPolyRespHandler(fd)
       int             fd;
  {
    /* NULL -- recvd msg got to sesmgr */
    sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPOLY);
    showCollabInfo(pFrontAppData->sbMsgBuf);
    return 0;
  }

  /*
   * Function
   */
```

```
int
collRecvdMsgShmPolyInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmPolyInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmPolyInHandler()->couldn't shMemDelete!\
        n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGSHMPOLY
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collStartPntrReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_PNTR, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartPntrRespHandler(fd)
     int              fd;
{
  /* start a pntr comm infrastructure.. one pntr wid per member */
  /* create and popup pntr comm controller */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (pntrStartFunc != NULL) {
    (*pntrStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"pntrStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_PNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndPntrReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_PNTR, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndPntrRespHandler(fd)
     int             fd;
{
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;
  /* terminate a pntr comm channel destroy wids etc */
  /* destroy popdown pntr comm controller */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (pntrEndFunc != NULL) {
    (*pntrEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"pntrEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_PNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendPntrReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_PNTR, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendPntrRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPntrRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (pntrSendFileFunc != NULL) {
    (*pntrSendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"pntrSendFileFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_PNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendPntrInHandler(fd)
     int             fd;
{
  /* recv msg from outside.. update local view */
  hostData       *pHostColl;
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPntrInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_PNTR);
  if (pntrRecvFileFunc != NULL) {
    (*pntrRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"pntrRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPntrReq(pHostColl, pPntrD)
     hostData       *pHostColl;
     shaDoubles     *pPntrD;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORPNTR
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = strlen(str) + 1;
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgPntrReq()->couldn't shMemReuseSegment!\n"
          );
    }
    memcpy(pShmInfo->shmAddr, str, n);
    collSendMsgShmPntrReq(pHostColl, pShmInfo);
    return 0;
  }
#endif              /* USESHAREDMEMFORPNTR */

  checkConn();
  sendReqString(REQ_SEND_MSGPNTR, NULL);
  PntrBiteOut(pHostColl->fdSocket, pPntrD);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPntrRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPntrRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (pntrSendMsgFunc != NULL) {
    (*pntrSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"pntrSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPntrInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  static shaDoubles pntrData;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPntrInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  PntrBiteIn(fd, &pntrData);
  if (pntrRecvMsgFunc != NULL) {
    (*pntrRecvMsgFunc) (pHostColl, &senderSIdTag, &pntrData);
  }
  else{
    fprintf(stderr,"pntrRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPntrReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_RECVD_MSGPNTR, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPntrRespHandler(fd)
     int             fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgPntrInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgPntrInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPntrReq(pHostColl, pShmInfo)
     hostData        *pHostColl;
     shmInfo         *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmPntrReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMPNTR, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPntrRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmPntrRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (pntrSendMsgFunc != NULL) {
    (*pntrSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"pntrSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPntrInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  static shaDoubles pntrData;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmPntrInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmPntrInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
    fprintf(stderr, "collSendMsgShmPntrInHandler()->SHM recon problem\n");
    return -1;
  }
  buf = pShmInfo->shmAddr;
  if (pntrRecvMsgFunc != NULL) {
    (*pntrRecvMsgFunc) (pHostColl, &senderSIdTag, &pntrData);
  }
  else{
    fprintf(stderr,"pntrRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGSHMPNTR
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmPntrReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmPntrReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMPNTR, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmPntrRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPNTR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmPntrInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmPntrInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmPntrInHandler()->couldn't shMemDelete!\
        n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGSHMPNTR
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
```

```
}



/*
 * Function
 */
int
collStartCursorReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_CURSOR, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartCursorRespHandler(fd)
     int              fd;
{
  /* start a cursor comm infrastructure.. one cursor wid per member */
  /* create and popup cursor comm controller */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (cursorStartFunc != NULL) {
    (*cursorStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"cursorStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_CURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndCursorReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_CURSOR, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
```

```
}

/*
 * Function
 */
int
collEndCursorRespHandler(fd)
     int              fd;
{
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;
  /* terminate a cursor comm channel destroy wids etc */
  /* destroy popdown cursor comm controller */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (cursorEndFunc != NULL) {
    (*cursorEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"cursorEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_CURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendCursorReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_CURSOR, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendCursorRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendCursorRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (cursorSendFileFunc != NULL) {
    (*cursorSendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"cursorSendFileFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_CURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendCursorInHandler(fd)
     int              fd;
{
  /* recv msg from outside.. update local view */
  hostData      *pHostColl;
  char          *buf;
  shastraIdTag   senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendCursorInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_CURSOR);
  if (cursorRecvFileFunc != NULL) {
    (*cursorRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"cursorRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgCursorReq(pHostColl, pCursorD)
     hostData       *pHostColl;
     shaDoubles     *pCursorD;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORCURSOR
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = strlen(str) + 1;
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgCursorReq()->couldn't shMemReuseSegment!\
          n");
    }
    memcpy(pShmInfo->shmAddr, str, n);
    collSendMsgShmCursorReq(pHostColl, pShmInfo);
    return 0;
  }
#endif                 /* USESHAREDMEMFORCURSOR */

  checkConn();
  sendReqString(REQ_SEND_MSGCURSOR, NULL);
  CursorBiteOut(pHostColl->fdSocket, pCursorD);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgCursorRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgCursorRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (cursorSendMsgFunc != NULL) {
    (*cursorSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"cursorSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
```

```
 * Function
 */
int
collSendMsgCursorInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  static shaDoubles cursorData;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgCursorInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  CursorBiteIn(fd, &cursorData);
  if (cursorRecvMsgFunc != NULL) {
    (*cursorRecvMsgFunc) (pHostColl, &senderSIdTag, &cursorData);
  }
  else{
    fprintf(stderr,"cursorRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGCURSOR)
      ;
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgCursorReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_RECVD_MSGCURSOR, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgCursorRespHandler(fd)
     int             fd;
```

```
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgCursorInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgCursorInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGCURSOR)
      ;
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmCursorReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmCursorReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMCURSOR, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
```

```
}

/*
 * Function
 */
int
collSendMsgShmCursorRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmCursorRespHandler()->NULL Host data!\n")
        ;
    return -1;
  }

  if (cursorSendMsgFunc != NULL) {
    (*cursorSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"cursorSendMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmCursorInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  static shaDoubles cursorData;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmCursorInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmCursorInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
    fprintf(stderr, "collSendMsgShmCursorInHandler()->SHM recon problem\n")
        ;
    return -1;
  }
  buf = pShmInfo->shmAddr;
  if (cursorRecvMsgFunc != NULL) {
    (*cursorRecvMsgFunc) (pHostColl, &senderSIdTag, &cursorData);
  }
  else{
    fprintf(stderr,"cursorRecvMsgFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_SEND_MSGSHMCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmCursorReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmCursorReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMCURSOR, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmCursorRespHandler(fd)
     int             fd;
{
  /* NULL -- recvd msg got to sesmgr */
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
```

```
}

/*
 * Function
 */
int
collRecvdMsgShmCursorInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmCursorInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmCursorInHandler()->couldn't shMemDelete
        !\n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n",
      REQ_RECVD_MSGSHMCURSOR);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
collStartXSCntlReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_XSCNTL, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartXSCntlRespHandler(fd)
```

```
     int             fd;
{
  /* start a xsCntl comm infrastructure.. */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collStartXSCntlRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  if (xsCntlStartFunc != NULL) {
    (*xsCntlStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"xsCntlStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_XSCNTL);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndXSCntlReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_XSCNTL, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndXSCntlRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  /* terminate a xsCntl comm channel */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (xsCntlEndFunc != NULL) {
    (*xsCntlEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"xsCntlEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_XSCNTL);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendXSCntlReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_XSCNTL, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendXSCntlRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendXSCntlRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (xsCntlSendFileFunc != NULL) {
    (*xsCntlSendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"xsCntlSendFileFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_XSCNTL);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
```

```
int
collSendXSCntlInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendXSCntlInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  if (xsCntlRecvFileFunc != NULL) {
    (*xsCntlRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"xsCntlRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);

  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_XSCNTL);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgXSCntlReq(pHostColl, pXSCBites)
     hostData       *pHostColl;
     xsCntlDatas    *pXSCBites;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORXSCD
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = 0;      /* HMMM */
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgXSCntlReq()->couldn't shMemReuseSegment!\
          n");
    }
    xsCntlDatasMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pXSCBites);
    collSendMsgShmXSCntlReq(pHostColl, pShmInfo);
    return 0;
  }
#endif              /* USESHAREDMEMFORXSCD */

  checkConn();
  sendReqString(REQ_SEND_MSGXSCNTL, NULL);
  XSCntlBitesOut(pHostColl->fdSocket, pXSCBites);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgXSCntlRespHandler(fd)
     int            fd;
{
  hostData      *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgXSCntlRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (xsCntlSendMsgFunc != NULL) {
    (*xsCntlSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"xsCntlSendMsgFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGXSCNTL);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendMsgXSCntlInHandler(fd)
     int            fd;
{
  hostData      *pHostColl;
  /* recv msg from outside.. update local view */
  char          *buf;
  shastraIdTag    senderSIdTag;
  static xsCntlDatas xsCntlBites;

  pHostColl = mplexGetHostData(fd);
```

```
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendMsgXSCntlInHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);

    XSCntlBitesIn(fd, &xsCntlBites);
    if (xsCntlRecvMsgFunc != NULL) {
      (*xsCntlRecvMsgFunc) (pHostColl, &senderSIdTag, &xsCntlBites);
    }
    else{
      fprintf(stderr,"xsCntlRecvMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
         REQ_SEND_MSGXSCNTL);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collRecvdMsgXSCntlReq(pHostColl, nameBuf)
       hostData       *pHostColl;
       char           *nameBuf;
  {
    checkConn();
    sendReqString(REQ_RECVD_MSGXSCNTL, NULL);
    sendDataString(nameBuf);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collRecvdMsgXSCntlRespHandler(fd)
       int             fd;
  {
    /* NULL -- recvd msg got to sesmgr */
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGXSCNTL);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
```

```
 * Function
 */
int
collRecvdMsgXSCntlInHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgXSCntlInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGXSCNTL)
      ;
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmXSCntlReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmXSCntlReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMXSCNTL, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmXSCntlRespHandler(fd)
     int             fd;
```

```
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmXSCntlRespHandler()->NULL Host data!\n")
        ;
    return -1;
  }

  if (xsCntlSendMsgFunc != NULL) {
    (*xsCntlSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"xsCntlSendMsgFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMXSCNTL
       );
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendMsgShmXSCntlInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  shastraIdTag    senderSIdTag;
  int             shmId;
  shmInfo        *pShmInfo;
  static xsCntlDatas xsCntlBites;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgShmXSCntlInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);

  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmXSCntlInHandler()->no non-local SHM\n");
    return -1;
  }
  pShmInfo = mplexInShmInfo(fd);
  if (!shMemReconnect(pShmInfo, shmId)) {
        fprintf(stderr, "collSendMsgShmXSCntlInHandler()->SHM recon problem\n")
            ;
        return -1;
      }
      xsCntlDatasMemIn(pShmInfo->shmAddr, pShmInfo->shmSize, &xsCntlBites);
      if (xsCntlRecvMsgFunc != NULL) {
        (*xsCntlRecvMsgFunc) (pHostColl, &senderSIdTag, &xsCntlBites);
      }
      else{
        fprintf(stderr,"xsCntlRecvMsgFunc()->no handler!\n");
      }
      /*
       * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
           REQ_SEND_MSGSHMXSCNTL);
       * showCollabInfo(pFrontAppData->sbMsgBuf);
       */
      return 0;
    }

    /*
     * Function
     */
    int
    collRecvdMsgShmXSCntlReq(pHostColl, pShmInfo)
         hostData        *pHostColl;
         shmInfo         *pShmInfo;
    {
      if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
        fprintf(stderr, "collRecvdMsgShmXSCntlReq()->no non-local SHM\n");
        return -1;
      }
      checkConn();
      sendReqString(REQ_RECVD_MSGSHMXSCNTL, NULL);
      ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
      cmFlush(pHostColl->fdSocket);
      return 0;
    }

    /*
     * Function
     */
    int
    collRecvdMsgShmXSCntlRespHandler(fd)
         int              fd;
    {
      /* NULL -- recvd msg got to sesmgr */
      return 0;
      /*
       * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n",
           REQ_RECVD_MSGSHMXSCNTL);
       * showCollabInfo(pFrontAppData->sbMsgBuf);
       */
    }
```

```
/*
 * Function
 */
int
collRecvdMsgShmXSCntlInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  shmInfo        *pShmInfo;
  int             shmId;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmXSCntlInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmXSCntlInHandler()->couldn't shMemDelete
        !\n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n",
      REQ_RECVD_MSGSHMXSCNTL);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collStartPictReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_START_PICT, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collStartPictRespHandler(fd)
     int              fd;
{
  /* start a pict comm infrastructure.. */
  shastraIdTag    senderSIdTag;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collStartPictRespHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  if (pictStartFunc != NULL) {
    (*pictStartFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"pictStartFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_PICT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collEndPictReq(pHostColl)
     hostData       *pHostColl;
{
  checkConn();
  sendReqString(REQ_END_PICT, NULL);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collEndPictRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  /* terminate a pict comm channel */
  pHostColl = mplexGetHostData(fd);
  ShastraIdTagIn(fd, &senderSIdTag);
  if (pictEndFunc != NULL) {
    (*pictEndFunc) (pHostColl, &senderSIdTag);
  }
  else{
    fprintf(stderr,"pictEndFunc()->no handler!\n");
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_PICT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendPictReq(pHostColl, nameBuf)
     hostData       *pHostColl;
     char           *nameBuf;
{
  checkConn();
  sendReqString(REQ_SEND_PICT, NULL);
  sendDataString(nameBuf);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendPictRespHandler(fd)
     int             fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPictRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (pictSendFileFunc != NULL) {
    (*pictSendFileFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"pictSendFileFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_PICT);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendPictInHandler(fd)
```

```
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendPictInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);

  buf = cmReceiveString(fd);
  if (pictRecvFileFunc != NULL) {
    (*pictRecvFileFunc) (pHostColl, &senderSIdTag, buf);
  }
  else{
    fprintf(stderr,"pictRecvFileFunc()->no handler!\n");
  }
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(buf);

  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_PICT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPictReq(pHostColl, pPCBites)
     hostData       *pHostColl;
     pictPieces     *pPCBites;
{
  shmInfo        *pShmInfo;
  int             n;

#ifdef USESHAREDMEMFORPICT
  if (pFrontSId->lIPAddr == pHostColl->pSId->lIPAddr) {
    pShmInfo = mplexOutShmInfo(pHostColl->fdSocket);
    n = 0;      /* HMMM */
    if (shMemReuseSegment(pShmInfo, ((n > 10240) ? n : 10240)) == 0) {
      fprintf(stderr, "collSendMsgPictReq()->couldn't shMemReuseSegment!\n"
          );
    }
    pictPiecesMemOut(pShmInfo->shmAddr, pShmInfo->shmSize, pPCBites);
    collSendMsgShmPictReq(pHostColl, pShmInfo);
    return 0;
  }
```

```
#endif              /* USESHAREDMEMFORPICT */

  checkConn();
  sendReqString(REQ_SEND_MSGPICT, NULL);
  PictDataBitesOut(pHostColl->fdSocket, pPCBites);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgPictRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPictRespHandler()->NULL Host data!\n");
    return -1;
  }

  if (pictSendMsgFunc != NULL) {
    (*pictSendMsgFunc) (pHostColl);
  }
  else{
    fprintf(stderr,"pictSendMsgFunc()->no handler!\n");
  }
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGPICT);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collSendMsgPictInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  /* recv msg from outside.. update local view */
  char           *buf;
  shastraIdTag    senderSIdTag;
  static pictPieces pictBites;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collSendMsgPictInHandler()->NULL Host data!\n");
```

```
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);

    PictDataBitesIn(fd, &pictBites);
    if (pictRecvMsgFunc != NULL) {
      (*pictRecvMsgFunc) (pHostColl, &senderSIdTag, &pictBites);
    }
    else{
      fprintf(stderr,"pictRecvMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_SEND_MSGPICT
        );
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collRecvdMsgPictReq(pHostColl, nameBuf)
       hostData       *pHostColl;
       char           *nameBuf;
  {
    checkConn();
    sendReqString(REQ_RECVD_MSGPICT, NULL);
    sendDataString(nameBuf);
    cmFlush(pHostColl->fdSocket);
    return 0;
  }

  /*
   * Function
   */
  int
  collRecvdMsgPictRespHandler(fd)
       int              fd;
  {
    /* NULL -- recvd msg got to sesmgr */
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGPICT);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
```

```
int
collRecvdMsgPictInHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
  char           *nameBuf;
  shastraIdTag    senderSIdTag;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgPictInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  /* recvd ack that all collabs have heard, delete local buf */
  nameBuf = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGPICT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  printf("deleting %s\n", nameBuf);
  /* is a tmp file */
  free(nameBuf);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPictReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collSendMsgShmPictReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SEND_MSGSHMPICT, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collSendMsgShmPictRespHandler(fd)
     int              fd;
{
  hostData       *pHostColl;
```

```
    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendMsgShmPictRespHandler()->NULL Host data!\n");
      return -1;
    }

    if (pictSendMsgFunc != NULL) {
      (*pictSendMsgFunc) (pHostColl);
    }
    else{
      fprintf(stderr,"pictSendMsgFunc()->no handler!\n");
    }
    return 0;
    /*
     * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SEND_MSGSHMPICT);
     * showCollabInfo(pFrontAppData->sbMsgBuf);
     */
  }

  /*
   * Function
   */
  int
  collSendMsgShmPictInHandler(fd)
       int              fd;
  {
    hostData       *pHostColl;
    /* recv msg from outside.. update local view */
    shastraIdTag    senderSIdTag;
    int             shmId;
    shmInfo        *pShmInfo;
    static pictPieces pictBites;

    pHostColl = mplexGetHostData(fd);
    if (pHostColl == NULL) {
      fprintf(stderr, "collSendMsgShmPictInHandler()->NULL Host data!\n");
      return -1;
    }

    ShastraIdTagIn(fd, &senderSIdTag);
    ShastraIntIn(fd, &shmId);

    if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
      fprintf(stderr, "collSendMsgShmPictInHandler()->no non-local SHM\n");
      return -1;
    }
    pShmInfo = mplexInShmInfo(fd);
    if (!shMemReconnect(pShmInfo, shmId)) {
      fprintf(stderr, "collSendMsgShmPictInHandler()->SHM recon problem\n");
      return -1;
    }
    pictPiecesMemIn(pShmInfo->shmAddr, pShmInfo->shmSize, &pictBites);
    if (pictRecvMsgFunc != NULL) {
```

```
    (*pictRecvMsgFunc) (pHostColl, &senderSIdTag, &pictBites);
  }
  else{
    fprintf(stderr,"pictRecvMsgFunc()->no handler!\n");
  }
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_SEND_MSGSHMPICT);
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmPictReq(pHostColl, pShmInfo)
     hostData       *pHostColl;
     shmInfo        *pShmInfo;
{
  if (pFrontSId->lIPAddr != pHostColl->pSId->lIPAddr) {
    fprintf(stderr, "collRecvdMsgShmPictReq()->no non-local SHM\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_RECVD_MSGSHMPICT, NULL);
  ShastraIntOut(pHostColl->fdSocket, &pShmInfo->shmId);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collRecvdMsgShmPictRespHandler(fd)
     int              fd;
{
  /* NULL -- recvd msg got to sesmgr */
  return 0;
  /*
   * sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_RECVD_MSGSHMPICT)
      ;
   * showCollabInfo(pFrontAppData->sbMsgBuf);
   */
}

/*
 * Function
 */
int
collRecvdMsgShmPictInHandler(fd)
```

```
     int             fd;
{
  hostData       *pHostColl;
  shastraIdTag    senderSIdTag;
  shmInfo        *pShmInfo;
  int             shmId;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collRecvdMsgShmPictInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &senderSIdTag);
  ShastraIntIn(fd, &shmId);
  pShmInfo = mplexOutShmInfo(fd);
  if (shMemDelete(pShmInfo, shmId) == 0) {
    fprintf(stderr, "collRecvdMsgShmPictInHandler()->couldn't shMemDelete!\
        n");
  }
  /* recvd ack that all collabs have heard, delete shared seg */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in)-- %s\n", REQ_RECVD_MSGSHMPICT
      );
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTextReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGTEXT, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTextRespHandler(fd)
```

```
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTextInHandler(fd)
     int             fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgTextInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabCommRecvdMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGTEXT);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTextFileReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGTEXTFILE, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTxtFileRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXTFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgTxtFileInHandler(fd)
     int             fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgTxtFileInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  /* show in dialog */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGTEXTFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
```

```
int
collCommMsgAudioReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGAUDIO, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collCommMsgAudioRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgAudioInHandler(fd)
     int             fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgAudioInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  /* send to service tool for handling */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGAUDIO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int
collCommMsgAudioFileReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGAUDIOFILE, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collCommMsgAudioFileRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIOFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgAudioFileInHandler(fd)
     int             fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgAudioFileInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  /* send to service tool for handling */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGAUDIOFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int
collCommMsgVideoReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGVIDEO, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
 * Function
 */
int
collCommMsgVideoRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}
```

```
/*
 * Function
 */
int
collCommMsgVideoInHandler(fd)
     int              fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgVideoInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  /* send to service tool for handling */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGVIDEO);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int
collCommMsgVideoFileReq(pHostColl, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData       *pHostColl;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pToSIdTag;
     shastraIdTag   *pSIdTag;
     char           *sbMsg;
{
  checkConn();
  sendReqString(REQ_COMM_MSGVIDEOFILE, NULL);
  ShastraIdTagOut(pHostColl->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostColl->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostColl->fdSocket);
  return 0;
}

/*
```

```
 * Function
 */
int
collCommMsgVideoFileRespHandler(fd)
     int              fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEOFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collCommMsgVideoFileInHandler(fd)
     int              fd;
{
  /* receive sesm idtag, display recvd message */
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char           *sMsg;
  hostData       *pHostColl;

  pHostColl = mplexGetHostData(fd);
  if (pHostColl == NULL) {
    fprintf(stderr, "collCommMsgVideoFileInHandler()->NULL Host data!\n");
    return -1;
  }

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  /* send to service tool for handling */
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGVIDEOFILE);
  showCollabInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/*
 * clSvrCntl.c
 */
#include <stdio.h>
#include <string.h>

#include <shastra/shastra.h>

#include <shastra/utils/list.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/genui.h>

#include <shastra/shautils/clientHosts.h>
#include <shastra/shautils/kernelFronts.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>

#include <shastra/front/front.h>
#include <shastra/front/frontP.h>
#include <shastra/front/front_client.h>
#include <shastra/front/clSvrCntl.h>
#include <shastra/front/clSvrCntlP.h>
#include <shastra/front/shastraCntl.h>

static ShastraToolMode iClSvrModeMine;
static ShastraToolMode iClSvrMode;
static shastraId defServerSId = { NULL, NULL, TEST_SERVICE_NAME};
extern chooseOne     *pcoClSvr;
hostData      *pHostShaCurrClnt;
static shastraIdTag currClntSIdTag;


void
clSvrSetSelfModeOprn()
{
  iClSvrModeMine = shastraNameToMode(pFrontSId->nmApplicn);
}


char          **
getServerNameList(pSId)
    shastraId* pSId;
{
  char          **sbNames;

  if(pSId == NULL){
    if(iClSvrMode == 0){
      defServerSId.nmApplicn = pFrontSId->nmApplicn;
    }
    else{
      defServerSId.nmApplicn = shastraModeToName(iClSvrMode);
    }
    sbNames = clHosts2StrTab(&defServerSId, PSIDNMHOST | PSIDNMAPPL);
  }
  else{
    sbNames = clHosts2StrTab(pSId, PSIDNMHOST | PSIDNMAPPL);
  }
  return sbNames;
}

char          **
getServerNameListByService(iService)
     int iService;
{
  char          **sbNames;

  defServerSId.nmApplicn = shastraServiceToName(iService);
  sbNames = clHosts2StrTab(&defServerSId, PSIDNMHOST | PSIDNMAPPL);
  return sbNames;
}

void
setClSvrServerNamesOprn(pSId)
    shastraId *pSId;
{
  char          **sbNames, *sService;

  if(pcoClSvr == NULL){
    return;
    }
    sService = shastraModeToName(iClSvrMode);
    if(strcmp(pSId->nmApplicn,sService)){
      return; /*not current service type*/
    }
    sbNames = getServerNameList(pSId);
    chooseOneChangeList(pcoClSvr, sbNames, coNoInitialHighlight);
    if (sbNames) {
      strListDestroy(sbNames);
    }
  }

  /*
   * Function
   */
  void
  clSvrSetCurrHostOprn(pHost, fForce)
      hostData *pHost;
      int fForce;
  {
    if(!fForce && (pHostShaCurrClnt != NULL)){
      return; /*only set if not already set*/
    }
    pHostShaCurrClnt = pHost;
    if(pHostShaCurrClnt != NULL){
      currClntSIdTag = pHostShaCurrClnt->lSIDTag;
  #ifdef DEBUG
      fprintf(stderr,"currClntSIdTag = %ld, pHost = %ld\n",
          currClntSIdTag, pHost);
  #endif /* DEBUG */
    }
    else{
      clSvrUnselectOprn();
    }
  /*
    set and update user interface element flags.. mode etc
    */
  }

  /*
   * Function
   */
  void
  clSvrResetCurrHostOprn(pHost, fForce)
      hostData *pHost;
      int fForce;
  {
    if(!fForce && (pHostShaCurrClnt != pHost)){
      return;         /*only set if not already set*/
    }
    else{
      clSvrUnselectOprn();
    }
```

```
}

hostData *
clSvrHostFromService(iService, iClSvr)
     int iService;
     int iClSvr;
{
  hostData       *pHost;

  defServerSId.nmApplicn = shastraServiceToName(iService);
  pHost = getClntHostByIndex(&defServerSId, iClSvr);

  return pHost;
}

hostData *
getClSvrHostFromIndex(iClSvr)
    int              iClSvr;
{
  hostData       *pHost;
  shastraId      *pSId = NULL;

  if(currClntSIdTag){
    pSId = mapSIdTag2SId(&currClntSIdTag);
  }
  if(pSId == NULL){
    pSId = &defServerSId;
    defServerSId.nmApplicn = shastraModeToName(iClSvrMode);
  }
  pHost = getClntHostByIndex(pSId, iClSvr);
#ifdef DEBUG
  fprintf(stderr,"getClSVrHostFromIndex()->smIdTag = %ld, pHost = %ld\n",
      pHost->lSIDTag, pHost);
#endif /* DEBUG */
  return pHost;
}

void
clSvrSetModeOprn(iMode)
    ShastraToolMode             iMode;
{
  iClSvrMode = iMode;
  /*update the shown set*/
  defServerSId.nmApplicn = shastraModeToName(iClSvrMode);

  setClSvrServerNamesOprn(&defServerSId);
}

/*
 * Function
 */
void
clSvrUnselectOprn()
```

```
{
  pHostShaCurrClnt = NULL;
  currClntSIdTag = 0;
}

/*
 * Function
 */
void
clSvrSelectOprn(i)
    int             i;
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  clSvrSetCurrHostOprn(pHost, True);
  if (clientSelectFunc != NULL) {
    (*clientSelectFunc) (pHostShaCurrClnt);
  }
}


/*
 * Function
 */
void
clSvrRenameOprn(i, name)
    int             i;
    char *name;
{
  /*change*/
}

/*
 * Function
 */
void
clSvrDisconnectOprn(i)
    int             i;
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  if(clntTerminateReq(NULL, pHost) == -1){
    clSvrUtilPopupMessage("clntTerminateReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
clSvrTerminateOprn(i)
    int             i;
```

```
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  if(clntTerminateReq(NULL, pHost) == -1){
    clSvrUtilPopupMessage("clntTerminateReq() Error!\n");
    return;
  }
  clSvrUtilPopupMessage("This operation is presently disabled!\n");
}

/*
 * Function
 */
void
clSvrCreateOprn(sbName)
    char *sbName;
{
  printf("create %s on %s\n", shastraModeToName(iClSvrMode), sbName);
  /*execute a starter script*/
}

/*
 * Function
 */
void
clSvrServerOprn(sbName, iPort)
    char *sbName;
    int iPort;
{
  shastraId sId;
  shaCmdData *pCmdData = NULL;

  if(!strcmp(pFrontSId->nmApplicn,sbName) &&
     (pFrontSId->iPort == iPort)){
    clSvrUtilPopupMessage("Warning: Connecting to self!\n");
  }
  memset(&sId, 0, sizeof(shastraId));
  sId.nmApplicn = shastraModeToName(iClSvrMode);
  sId.nmHost = sbName;
  sId.iPort = iPort;
/*CHECK*/
  sId.lSIDTag = mplexGetUniqueId();
  sId.lIPAddr = hostName2IPAddress(sbName);
  /*check if already connected*/
  if(getClntHostByIdTag(&sId, &sId.lSIDTag) != NULL){
    clSvrUtilPopupMessage("Warning: Already connected to host!\n");
  }
  printf("server connect to %s on %s\n", sId.nmApplicn, sbName);
  /* connect using non-shastra info */
  if(clientControlDataFunc){
    (*clientControlDataFunc)(shastraModeToService(iClSvrMode), &pCmdData);
    if(pCmdData == NULL){
      clSvrUtilPopupMessage("Invalid Control Data!\n");
```

```
        return;
      }
    }
    else{
      clSvrUtilPopupMessage("Can't Obtain Control Data!\n");
      return;
    }
    if(clntConnectReq(NULL, &sId, pCmdData) == -1){
      clSvrUtilPopupMessage("clntConnectReq() Error!\n");
      return;
    }
  }


  /*
   * Function
   */
  void
  clSvrConnectOprn(iWhich)
      int iWhich;
  {
    shastraIdTag *pSIdTag;
    shastraId *pSId;
    shaCmdData *pCmdData = NULL;

    pSIdTag = krFrNdx2SIdTag(iWhich);
    pSId = mapSIdTag2SId(pSIdTag);
    if(pSId == NULL){
      clSvrUtilPopupMessage("Invalid System!\n");
      return;
    }
    if(*pSIdTag == pFrontSId->lSIDTag){
      clSvrUtilPopupMessage("Warning: Connecting to self!\n");
    }
    /*check if already connected*/
    if(getClntHostByIdTag(pSId, pSIdTag) != NULL){
      clSvrUtilPopupMessage("Warning: Already connected!\n");
    }
    if(clientControlDataFunc){
      (*clientControlDataFunc)(shastraNameToService(pSId->nmApplicn), &
          pCmdData);
      if(pCmdData == NULL){
        clSvrUtilPopupMessage("Invalid Control Data!\n");
        return;
      }
    }
    else{
      clSvrUtilPopupMessage("Can't Obtain Control Data!\n");
      return;
    }
    if(clntConnectReq(NULL, pSId, pCmdData) == -1){
      clSvrUtilPopupMessage("clntConnectReq() Error!\n");
      return;
  }
}


void
clSvrOperationsOprn(pMgrCD, fUp)
    mgrCntlData *pMgrCD;
    int fUp;
{
  if(pHostShaCurrClnt == NULL){
    clSvrUtilPopupMessage("Invalid Current Server!\n");
    return;
  }
  if (clientOperatorFunc != NULL) {
    (*clientOperatorFunc) (pHostShaCurrClnt);
  }
}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#ifdef SHASTRA4SUN5
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/systeminfo.h>
#include <unistd.h>
int putenv(char *);
#endif
#include <sys/types.h>
#include <sys/socket.h>

#include <pwd.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <malloc.h>
#include <sys/errno.h>
#include <netdb.h>

#include <X11/Intrinsic.h>
#include <X11/Xutil.h>
#include <X11/StringDefs.h>

#include <Xm/Text.h>

#include <shastra/shastra.h>
#include <shastra/shastraStateDefs.h>

#include <shastra/utils/list.h>
```

```
#include <shastra/uitools/buttonBox.h>
#include <shastra/uitools/toggleBox.h>
#include <shastra/uitools/stateBox.h>
#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/chooseMany.h>
#include <shastra/uitools/callbackArg.h>
#include <shastra/uitools/dialog.h>
#include <shastra/uitools/confirmCB.h>
#include <shastra/uitools/miscUtils.h>
#include <shastra/uitools/strListUtilities.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>

#include <shastra/kernel/kernel.h>
#include <shastra/kernel/kernelMainCB.h>
#include <shastra/kernel/kernel_server.h>
#include <shastra/kernel/kernelFallback.h>
#include <shastra/kernel/kernel_client.h>
#include <shastra/kernel/kernelState.h>

#define SHASTRA_MALLOCDBGnn

static char *GetShastraBaseDir();
#ifdef SHASTRA4SUN5
extern char *strdup(Prot1(char *));
#endif

static shaKernelAppData kernelAppData;
shaKernelAppData *pKernelAppData = &kernelAppData;
static shastraId kernShastraId;
shastraId       *pKernelSId = &kernShastraId;

char            sbOutMsgBuf[1024];
#define DEBUG 0
int             debug = DEBUG;
extern int      errno;

int             fMainKernel;
int             fForcedXMainKernel;

void             getCmdLineArgs(Prot2(int , char ** ));
void             cmdLineUsage(Prot1(char **));
void             getRegisterInfo(Prot1(shastraId *));
void             uiCreate(Prot2(Widget, XtAppContext ));
int             shastraHandleXEvent();
int             kernelPortNum;
int             mainKernClntSocket;
unsigned long   kernelIPAddr;
int             iKernelFrontIndex;
#ifndef SHASTRA4SUN5
#define MAXNAMELEN 128
#endif
char             kernelHostName[MAXNAMELEN];
char             kernelUserName[MAXNAMELEN];
char             kernelHeadHostName[MAXNAMELEN];

shastraId        kernelShastraId;
shastraIds      *pShastraFrontIds;   /* fronts connected on kernel */

XtAppContext     shastraAppContext;

Widget           wgShastraTopLevel;

int              iXAppFileDes;
int              shastraServerStatus;

char            *shastraPasswd = SHASTRAPASSWORD;

char            *kernelAppName;
char            *kernelDispName;
char            *kernelPasswd;
int              kernelFNoGUI;

shaCmdData       serverCmdData;
cmCommand        serverCommandTab[] = SERVERCMDS;
#define NSERVERCMDS (sizeof(serverCommandTab)/sizeof(cmCommand))
/* number of commands */
int              serverNCmds = NSERVERCMDS;

#ifdef SHASTRA_MALLOCDBG
#ifdef SHASTRA4IRIS
#include<sys/types.h>
#include<malloc.h>
#elif defined SHASTRA4SUN4
int              malloc_debug(Prot1(int));
int              malloc_verify(Prot0(void));
#endif
#endif               /* SHASTRA_MALLOCDBG */

int              shastraFlush(Prot0(void));
int              shastraServiceSocket;
int              shastraPort;

shaCmdData       kernelCmdData;

cmCommand        kernelCmdTab[] = KERNEL_CLIENTCMDS;
```

```
#define KERNEL_NCMDS (sizeof(kernelCmdTab)/sizeof(cmCommand))
int             kernelNCmds = KERNEL_NCMDS;

cmCommand       kernelInCmdTab[] = KERNEL_CLIENTINCMDS;
#define KERNEL_INNCMDS (sizeof(kernelInCmdTab)/sizeof(cmCommand))
int             kernelInNCmds = KERNEL_INNCMDS;

hostData        hostMainKern;
hostData       *pHostMainKern = &hostMainKern;


void
shastraKernelSetupApplResDir()
{
  char sbName[1024], *sName;

  sName = resolveNameFromBase(pKernelAppData->sDirBase,
                  pKernelAppData->sDirDefs);
    sprintf(sbName,"XAPPLRESDIR=%s", sName);
    putenv(sbName);
}

int
main(argc, argv)
    int             argc;
    char          **argv;
{
        char *nname;
    FILE           *fpHome;
    char *sName;
    struct hostent *pHostEnt;
    extern int      closedChannelCleanUpHandler();
        uid_t auid;
        struct passwd *apass;
        struct linger soLinger;
        unsigned int temp;

    static XrmOptionDescRec xrmOptions[] = {
        DEFSHASTRAXRMOPTIONS
    };
    static XtResource xrmResources[] = {
     { XshaNbaseDirectory, XshaCbaseDirectory, XtRString, sizeof(String),
      XtOffsetOf(shaKernelAppData, sDirBase), XtRImmediate,
      (XtPointer)DEFSHASTRABASEDIR },
     { XshaNminimal, XshaCminimal, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaKernelAppData, fMinimal), XtRImmediate,
         (XtPointer)False },
     { XshaNconnect, XshaCconnect, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaKernelAppData, fConnect), XtRImmediate,
          (XtPointer)True },
     { XshaNnoGUI, XshaCnoGUI, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaKernelAppData, fNoGUI), XtRImmediate, (XtPointer)False}
          ,
  { XshaNusePixmap, XshaCusePixmap, XtRBoolean, sizeof(Boolean),
      XtOffsetOf(shaKernelAppData, fPixmap), XtRImmediate, (XtPointer)
          False},
  { XshaNhelp, XshaChelp, XtRBoolean, sizeof(Boolean),
   XtOffsetOf(shaKernelAppData, fHelp), XtRImmediate, (XtPointer)False }
       ,
  { XshaNservicePort, XshaCservicePort, XtRInt, sizeof(int),
   XtOffsetOf(shaKernelAppData, iSvcPort), XtRImmediate, (XtPointer)0 },
  { XshaNshastraPort, XshaCshastraPort, XtRInt, sizeof(int),
   XtOffsetOf(shaKernelAppData, iShaPort), XtRImmediate, (XtPointer)0 },
  { XshaNdebugLevel, XshaCdebugLevel, XtRInt, sizeof(int),
   XtOffsetOf(shaKernelAppData, iDbgLevel), XtRImmediate, (XtPointer)0 }
       ,
  { XshaNdefsDirectory, XshaCdefsDirectory, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sDirDefs), XtRImmediate,
   (XtPointer)DEFSHASTRADEFSDIR },
  { XshaNdataDirectory, XshaCdataDirectory, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sDirData), XtRImmediate,
   (XtPointer)DEFSHASTRADATADIR },
  { XshaNbinDirectory, XshaCbinDirectory, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sDirBin), XtRImmediate,
   (XtPointer)DEFSHASTRABINDIR },
  { XshaNlogFile, XshaClogFile, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sFileLog), XtRImmediate,
   (XtPointer)DEFSHASTRALOGFILE },
  { XshaNhomeFile, XshaChomeFile, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sFileHome), XtRImmediate,
   (XtPointer)DEFSHASTRAHOMEFILE },
  { XshaNappsFile, XshaCappsFile, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sFileApps), XtRImmediate,
   (XtPointer)DEFSHASTRAAPPSFILE },
  { XshaNusersFile, XshaCusersFile, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sFileUsers), XtRImmediate,
   (XtPointer)DEFSHASTRAUSERSFILE },
  { XshaNhostsFile, XshaChostsFile, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sFileHosts), XtRImmediate,
   (XtPointer)DEFSHASTRAHOSTSFILE },
  { XshaNlocalStarter, XshaClocalStarter, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sLocStart), XtRImmediate,
   (XtPointer)DEFSHASTRASTARTLOCAL },
  { XshaNremoteStarter, XshaCremoteStarter, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sRemStart), XtRImmediate,
   (XtPointer)DEFSHASTRASTARTREMOTE },
  { XshaNpassword, XshaCpassword, XtRString, sizeof(String),
   XtOffsetOf(shaKernelAppData, sPasswd), XtRImmediate,
   (XtPointer)DEFSHASTRAPASSWD },
 };

 xrmResources[0].default_addr = GetShastraBaseDir();

 wgShastraTopLevel = XtAppInitialize(&shastraAppContext, "ShastraKernel"
     ,
          xrmOptions, XtNumber(xrmOptions),
            &argc, argv, fallback_resources, NULL, 0);
    shastraAddConverters();

    XtVaGetApplicationResources(wgShastraTopLevel,
        (XtPointer)&kernelAppData,
        xrmResources, XtNumber(xrmResources),
        /*hardcoded non-overridable app resources vararg list*/
        XshaNhelp, False,
        XshaNusePixmap, False,
        NULL);
    /*sanity checking of resources*/

/*
    shastraKernelSetupApplResDir();
*/

    getCmdLineArgs(argc, argv);
    kernelAppName = KERNEL_SERVICE_NAME;/* store application name */
    if (kernelDispName == NULL) {
        kernelDispName = XDisplayName(NULL);
    }
    if (kernelPasswd == NULL) {
        kernelPasswd = SHASTRAPASSWORD;
    }
    mplexInit(wgShastraTopLevel,shastraAppContext);
    registerInit();
    kernFrontsInit();
    sesmFrontsInit();
    mplexRegisterErrHandler(closedChannelCleanUpHandler);

#ifdef SHASTRA4SUN5
    if (sysinfo(SI_HOSTNAME, kernelHostName, MAXNAMELEN) == -1) {
        perror("sysinfo()");
#else
    if (gethostname(kernelHostName, MAXNAMELEN) != 0) {
        perror("gethostname()");
#endif
        strcpy(kernelHostName, "anonymous.cs.purdue.edu");
    }
    if ((pHostEnt = gethostbyname(kernelHostName)) == NULL) {
        perror("gethostbyname()");
        return(0);
    }
        memcpy(&temp, &pHostEnt->h_addr_list[0][0], 4);
    kernelIPAddr = ntohl(temp);
    /*
     * printf("name : %s\n",kernelHostName);
     */
/* this used to read the host name from a file */
#ifdef ANCIENTUGLYCODE
    sName = resolveNameFrom2Bases(pKernelAppData->sDirBase,
        pKernelAppData->sDirDefs, pKernelAppData->sFileHome);
    if ((fpHome = fopen(sName, "r")) == NULL) {
        perror("fopen()");
        fprintf(stderr, "main()->couldn't open %s! Aborting..\n",
            sName);
        exit(-1);
    }
    fgets(kernelHeadHostName, MAXNAMELEN, fpHome);
    fclose(fpHome);
#endif
        nname = (char *)MasterKernelName(kernelHostName);
        if (nname)
        {
            strcpy(kernelHeadHostName,nname);
        }
        else
        {
            strcpy(kernelHeadHostName,kernelHostName);
        }
    /*kernelHeadHostName[strlen(kernelHeadHostName) - 1] = '\0';*/
    printf("name : %s\n", kernelHeadHostName);
    fForcedXMainKernel = 0;
    if (!strcmp(kernelHostName, kernelHeadHostName))
        {
                /* head?? */
        fMainKernel = 1;
    }
        else
        {
        fMainKernel = 0;
    }
        auid = getuid();
        apass = getpwuid(auid);
    strcpy(kernelUserName,apass->pw_name);  /* store user name */

    serverCmdData.pCmdTab = serverCommandTab;
    serverCmdData.nCmds = serverNCmds;
    serverCmdData.pCmdTabIn = NULL;
    serverCmdData.nCmdsIn = 0;

    if ((shastraServerStatus =
         cmOpenServerSocket(SHASTRA_SERVICE_NAME, 0, &serverCmdData,
                &shastraServiceSocket, NULL)) == -1) {
        /* OpenServerSocket registers the handler */
        fprintf(stderr, "main()->Server Start-up error!\n Quitting!\n");
        exit(-1);
    }
        soLinger.l_onoff = 0;
        soLinger.l_linger = 5;        /* seconds */
        setsockopt(shastraServiceSocket, SOL_SOCKET, SO_LINGER,
                   &soLinger,sizeof(struct linger));

    uiCreate(wgShastraTopLevel, shastraAppContext);
/*
    pMyKernelSId = getMyKernelShastraId();
```

```
        pMyKernelAD = getMyKernelAppData();
    */

        XFlush(XtDisplay(wgShastraTopLevel));
    /*
        iXAppFileDes = ConnectionNumber(XtDisplay(wgShastraTopLevel));

        if (mplexRegisterChannel(iXAppFileDes, shastraHandleXEvent,
                                     NULL, NULL) == -1) {
            fprintf(stderr, "main()->Couldn't register X Handler!\n");
        }
    */
        shastraPort = shastraServerStatus;

        /* connect to main kernel */
        if (!fMainKernel)
            {   /* only non-heads */
            kernelPortNum = cmClientConnect2Server(kernelHeadHostName,
                      SHASTRA_SERVICE_NAME, 0, &mainKernClntSocket);
            if (kernelPortNum == -1) {
            perror("cmClientConnect2Server()");
            }
            if ((kernelPortNum == -1) && (errno == ECONNREFUSED))
                {
            /* problem.. maybe no kernel */
            fprintf(stderr, "main()->couldn't register with master kernel!\n");
            fprintf(stderr, "main()->becoming a master kernel!\n");
            fMainKernel = 1;
            fForcedXMainKernel = 1;
            /* save name in file */
            sName = resolveNameFrom2Bases(pKernelAppData->sDirBase,
                pKernelAppData->sDirDefs, pKernelAppData->sFileHome);
            if ((fpHome = fopen(sName, "w")) == NULL)
                    {
                perror("fopen()");
                fprintf(stderr, "main()->couldn't open %s! Aborting..\n",
                        sName);
                exit(-1);
            }
            fprintf(fpHome, "%s\n", kernelHostName);
            fclose(fpHome);
            strcpy(kernelHeadHostName, kernelHostName);
            /*
             * should we try a loop-start main kernel here as
             * well?
             */
            }
                    else
                    {
                kernelCmdData.pCmdTab = kernelCmdTab;
                kernelCmdData.nCmds = kernelNCmds;
                kernelCmdData.pCmdTabIn = kernelInCmdTab;
                kernelCmdData.nCmdsIn = kernelInNCmds;
            pHostMainKern->fdSocket = mainKernClntSocket;
            pHostMainKern->sendList = listMakeNew();
            pHostMainKern->recvList = listMakeNew();
            pHostMainKern->fStatus = shaWait2Send;

            /* register handler */
        if (mplexRegisterChannel(pHostMainKern->fdSocket,
                                         shaClientHandler,
                     &kernelCmdData, NULL) == -1)
                {
            fprintf(stderr,
                             "main()->Couldn't Register Client Handler!!\n"
                                 );
                pHostMainKern->fStatus = shaError;
                return(0);
        }
        mplexSetHostData(pHostMainKern->fdSocket, pHostMainKern);
        getRegisterInfo(&kernelShastraId);
        /* after connecting,setting up handler */
        setShaKernIdOprn(mainKernClntSocket);   /* register ID with
                                 * MainKernel */
        }
    }
    /*
     * this needs to follow the !fMainKernel part, as a kernel may need
     * to become a main kernel  if the main one isn't up already
     */
    if (fMainKernel)
        {
                /* put shastraId in my own table */
                SetupKernelNameServer(shastraAppContext,kernelHostName);
        kernelPortNum = shastraServerStatus;    /* from
                             * cmopenServerSocket() */
        getRegisterInfo(&kernelShastraId);
        copyId(&kernelShastraId, &localShaIdIn[shastraServiceSocket]);
        shaKernFlags[shastraServiceSocket] = SHAKERNEL;
        updateShaKernIds();
        if (rgsbShastraKern != NULL) {
            strListDestroy(rgsbShastraKern);
        }
        rgsbShastraKern = pSIds2StrTab(&shastraKernIds, PSIDNMHOST);
        chooseOneChangeList(pcoShastraKern, rgsbShastraKern,
                    coNoInitialHighlight);
    }
    /* identify front index */
    iKernelFrontIndex = locateKernFronts(&kernelShastraId);
    if (iKernelFrontIndex != -1) {
        fprintf(stderr, "main()->locateKernFronts() already has index %d!\
            n",
            iKernelFrontIndex);
    } else {
        iKernelFrontIndex = occupyKrFrFreeSlot(&kernelShastraId);
    }
    pShastraFrontIds = getKernFrontSIds(&kernelShastraId);
    /* initially empty fronts */
    pShastraFrontIds->shastraIds_len = 0;
    pShastraFrontIds->shastraIds_val =
        (shastraId_P *) malloc(mplexGetMaxChannels() * sizeof(shastraId_P))
            ;

    if (rgsbShastraFront != NULL) {
        strListDestroy(rgsbShastraFront);
    }
    rgsbShastraFront = pSIds2StrTab(pShastraFrontIds, PSIDNMHOST |
        PSIDNMAPPL);
    chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                coNoInitialHighlight);

    if (rgsbShastraSesMgr != NULL) {
        strListDestroy(rgsbShastraSesMgr);
    }
    rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds, PSIDNMHOST |
        PSIDNMAPPL);
    chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
                coNoInitialHighlight);

    shastraFlush();
    mplexSetTimeout(7200000L);  /* 2hrs */

    mplexMain(shastraFlush);
        return(0);
}

void
uiCreate(wgParent, xac)
    Widget          wgParent;
    XtAppContext    xac;
{
    Widget          wgMainCmdShell;

    pcbArgPopup->operation = NULL;
    strcpy(pcbArgPopup->msg, "Callback Arg Uninitialized\n");

    /* Do the one time initialization of the choose one object */
    chooseOneInit(xac);

    /* Create the three shell widgets and all of their child widgets */
    wgMainCmdShell = createMainCmdShell(wgParent);
    wgConfirmsShell = createConfirmsShell(wgParent);

    /* Pop up the three shell widgets */
    XtPopup(wgMainCmdShell, XtGrabNone);
}

/*
```

```
 * Function
 */
int
shastraHandleXEvent(xDescr, dummyArg)
    int             xDescr;
    char           *dummyArg;
{
    XEvent          xev, xevNext;

        fprintf(stderr, "Handle X Event!\n");
    while (XtAppPending(shastraAppContext)) {
        XtAppNextEvent(shastraAppContext, &xev);
        if (xev.type == MotionNotify) {
            while (XtAppPending(shastraAppContext)) {
                XtAppPeekEvent(shastraAppContext, &xevNext);
                if (xevNext.type != MotionNotify) {
                    break;
                }
                if (xevNext.xmotion.window != xev.xmotion.window) {
                    break;
                }
                XtAppNextEvent(shastraAppContext, &xev);
            }
            /* compress motion notify events to last one */
        }
        XtDispatchEvent(&xev);
    }
    return(0);
}

/*
 * Function
 */
int
shastraFlush()
{
    XFlush(XtDisplay(wgShastraTopLevel));
        return(0);
}

void
getRegisterInfo(pSId)
    shastraId      *pSId;
{
    double          load;
    extern void     getLoadAvg(Prot1(double *));

    memset(pSId, 0, sizeof(shastraId *));


    pSId->lIPAddr = kernelIPAddr;
    printf("%lu (%lx) -- %s\n", pSId->lIPAddr, pSId->lIPAddr,
           ipaddr2str(pSId->lIPAddr));
    pSId->lSIDTag = kernelIPAddr;   /* for kernels IPAddr is their tag */

    getLoadAvg(&load);
    printf("load is %f\n", load);
    pSId->dLoadAvg = load;

    pSId->nmHost = strdup(kernelHostName);
    pSId->nmDisplay = strdup(kernelDispName);
    pSId->nmApplicn = strdup(kernelAppName);
    pSId->nmUser = strdup(kernelUserName);
    pSId->webname = strdup(kernelUserName);
    pSId->nmPasswd = strdup(kernelPasswd);

    pSId->iPort = kernelPortNum;

    pSId->iProcId = getpid();

    if (debug) {
        outputId(stdout, pSId);
    }
}

/*
 * Function --
 */
void
showInfo(s)
    char            *s;
{
    static XmTextPosition currentPosn;
    outputTextToWidget(s, wgStatusText, &currentPosn);
    /*
     * fprintf(stdout, "%s", s);
     */
}

void
cmdLineUsage(argv)
    char           **argv;
{
    fprintf(stderr, "usage: %s [options]\n", argv[0]);
    fprintf(stderr, "  where options are:\n");
    fprintf(stderr, "    -display <display name>\n");
    fprintf(stderr, "    -help\n");
    fprintf(stderr, "    -nogui\n");
    fprintf(stderr, "    -passwd <password>\n");
}

void
getCmdLineArgs(argc, argv)
    int             argc;
    char           **argv;
```

```
{
    int             i;
    for (i = 1; i < argc; i++) {
        if (!strcmp("-display", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelDispName = argv[i];
            continue;
        }
        if (!strcmp("-help", argv[i])) {
            cmdLineUsage(argv);
        }
        if (!strcmp("-nogui", argv[i])) {
            kernelFNoGUI = 1;
            continue;
        }
        if (!strcmp("-passwd", argv[i])) {
            if (++i >= argc)
                cmdLineUsage(argv);
            kernelPasswd = argv[i];
            continue;
        }
        cmdLineUsage(argv);
    }
}

/*For static linking*/
#ifdef SHASTATIC
int dlopen() { return(0);}
int dlclose() {return(0); }
int dlsym() {return(0); }
#endif

static char *GetShastraBaseDir()
{
    char *dname;

    if (dname = getenv("SHASTRADIR"))
    {
         return(dname);
    }
    else
    {
        dname = strdup(DEFSHASTRABASEDIR);
    }
    return(dname);
}
```

```
/*************************************************************************
    ***/
/*************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*************************************************************************
    ***/
/*************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/strListUtilities.h>
#include <shastra/uitools/callbackArg.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/datacomm/shastraDataH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/kernel/kernel_server.h>
#include <shastra/kernel/kernel.h>
#include <shastra/kernel/kernelMainCB.h>
#include <shastra/kernel/kernel_client.h>

extern int      debug;

#define checkConn()         \
    if (pHostMainKern->fStatus == shaError) {               \
        fprintf(stderr,"Connection to shastra is bad!\n");  \
    }

#define sendReqString(s,arg)        \
    if(hostSendQueuedRequest(pHostMainKern, s, arg) == -1){ \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr,"Error in Sending Operation Request\n"); \
    }

#define sendDataString(s)       \
    if(cmSendString(pHostMainKern->fdSocket, s) == -1){ \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr,"Error in Sending Operation Data\n");    \
    }


#define ShastraIdIn(filedesc, pShaId)           \
    if(shastraIdIn(pHostMainKern->fdSocket, pShaId) == -1){ \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Receiving SID from Main\n"); \
    }

#define ShastraIdOut(filedesc, pShaId)          \
    if(shastraIdOut(pHostMainKern->fdSocket, pShaId) == -1){    \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Sending SID to Main\n"); \
    }

#define ShastraIdsIn(filedesc, pShaIds)         \
    if(shastraIdsIn(pHostMainKern->fdSocket, pShaIds) == -1){   \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Receiving SIDs from Main\n");    \
    }

#define ShastraIdsOut(filedesc, pShaIds)            \
    if(shastraIdsOut(pHostMainKern->fdSocket, pShaIds) == -1){  \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Sending SIDs to Main\n");    \
    }

#define ShastraIdTagIn(filedesc, pShaIdTag)         \
    if(shastraIdTagIn(pHostMainKern->fdSocket, pShaIdTag) == -1){   \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Receiving SIDTag from Main\n");  \
    }

#define ShastraIdTagOut(filedesc, pShaIdTag)            \
    if(shastraIdTagOut(pHostMainKern->fdSocket, pShaIdTag) == -1){  \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Sending SIDTag to Main\n");  \
    }

#define ShastraIdTagsIn(filedesc, pShaIdTags)           \
    if(shastraIdTagsIn(pHostMainKern->fdSocket, pShaIdTags) == -1){ \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Receiving SIDTags from Main\n"); \
        return(0);                                          \
    }

#define ShastraIdTagsOut(filedesc, pShaIdTags)          \
    if(shastraIdTagsOut(pHostMainKern->fdSocket, pShaIdTags) == -1){    \
        pHostMainKern->fStatus = shaError;                              \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        fprintf(stderr, "Error Sending SIDTags to Main\n"); \
    }

#define ShastraULongIn(filedesc, pULong)            \
    if(shaULongIn(pHostMainKern->fdSocket, pULong) == -1){    \
        pHostMainKern->fStatus = shaError;                                \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);
            \
        fprintf(stderr, "Error Receiving pULong from Main\n");    \
    }

#define ShastraULongOut(filedesc, pULong)           \
    if(shaULongOut(pHostMainKern->fdSocket, pULong) == -1){   \
        pHostMainKern->fStatus = shaError;                                \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);
            \
        fprintf(stderr, "Error Sending pULong to Main\n");    \
    }




/*
 * Function
 */
int
startSystemExportOprn(pSId)
    shastraId      *pSId;
{
    checkConn();
    sendReqString(REQ_START_SYSTEM, NULL);
    if (debug) {
        outputId(stderr, pSId);
    }
    ShastraIdOut(pHostMainKern->fdSocket, pSId);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}


/*
 * Function
 */
int
startSystemOprn(iObjIndex)
    int             iObjIndex;
{
    checkConn();
    sendReqString(REQ_START_SYSTEM, NULL);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
void
endSystemOprn(iObjIndex)
    int             iObjIndex;
{
    shastraIds     *pSIds;
    shastraId      *pSId;

    pSIds = (shastraIds *) pcbArgPopup->oprnAltArg;
    pSId = pSIds->shastraIds_val[iObjIndex];
    if (debug) {
        outputId(stdout, pSId);
    }
    if (strcmp(pcbArgPopup->argBuffer, pSId->nmPasswd)) {
        /* passwd mismatch */
        sprintf(sbOutMsgBuf, "Kill()->Password Incorrect -- Aborted\n");
        showInfo(sbOutMsgBuf);
        return;
    }
    if (pSId->lIPAddr != kernelShastraId.lIPAddr) { /* not local front */
        if (fMainKernel) {
            int             outFd;  /* non local sesm, send kill
                         * message */

            outFd = shaKernId2Fd(pSId); /* get fd of kern for
                             * this front */
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "Kill()->Unknown Kernel -- Aborted\n")
                    ;
                showInfo(sbOutMsgBuf);
                return;
            }
            putShaEndSysHandler(outFd, pSId);
        } else {
            checkConn();
            sendReqString(REQ_END_SYSTEM, NULL);
            ShastraIdOut(pHostMainKern->fdSocket, pSId);
        }
    } else {
        int             outFd;  /* local sesm, kill */
        outFd = shaFrontId2Fd(pSId);
        if (outFd == -1) {
            sprintf(sbOutMsgBuf, "Kill()->Unknown System -- Aborted\n");
            showInfo(sbOutMsgBuf);
            return;
        }
        putShaTerminateHandler(outFd);
    }
    cmFlush(pHostMainKern->fdSocket);
        return;
}

/*
 * Function
 */
void
endKernelOprn(iObjIndex)
    int             iObjIndex;
{
    shastraId      *pSId;

    if (iObjIndex < 0) {
        return;
    }
    pSId = shastraKernIds.shastraIds_val[iObjIndex];
    if (debug) {
        outputId(stdout, pSId);
    }
    if (strcmp(pcbArgPopup->argBuffer, pSId->nmPasswd)) {
        /* passwd mismatch */
        sprintf(sbOutMsgBuf, "KillKern()->Password Incorrect -- Aborted\n")
            ;
        showInfo(sbOutMsgBuf);
    }
    if (pSId->lIPAddr != kernelShastraId.lIPAddr) { /* not me */
        if (fMainKernel) {
            int             outFd;  /* non local sesm, send kill
                         * message */

            outFd = shaKernId2Fd(pSId); /* get fd of kern for
                             * this sesMgr */
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "KillKern()->Unknown Kernel -- Aborted
                    \n");
                showInfo(sbOutMsgBuf);
                return;
            }
            putShaTerminateHandler(outFd);
        } else {
            checkConn();
            sendReqString(REQ_END_SYSTEM, NULL);
            ShastraIdOut(pHostMainKern->fdSocket, pSId);
        }
    } else {
        quitOprn(0);
    }
    cmFlush(pHostMainKern->fdSocket);
        return;
}

/*
 * Function
 */
void
endSesMgrOprn(iObjIndex)
    int             iObjIndex;
{
    shastraId      *pSId;

    if (iObjIndex < 0) {
        return;
    }
    pSId = shastraSesmIds.shastraIds_val[iObjIndex];
    if (debug) {
        outputId(stdout, pSId);
    }
    if (strcmp(pcbArgPopup->argBuffer, pSId->nmPasswd)) {
        /* passwd mismatch */
        sprintf(sbOutMsgBuf, "KillSesm()->Password Incorrect -- Aborted\n")
            ;
        showInfo(sbOutMsgBuf);
        return;
    }
    if (pSId->lIPAddr != kernelShastraId.lIPAddr) { /* not local sesm */
        if (fMainKernel) {
            int             outFd;  /* non local sesm, send kill
                         * message */
            outFd = shaKernId2Fd(pSId); /* get fd of kern for
                             * this sesMgr */
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "KillSesm()->Unknown Kernel -- Aborted
                    \n");
                showInfo(sbOutMsgBuf);
                return;
            }
            putShaEndSysHandler(outFd, pSId);
        } else {
            checkConn();
```

```
                sendReqString(REQ_END_SYSTEM, NULL);
                ShastraIdOut(pHostMainKern->fdSocket, pSId);
            }
        } else {
            int             outFd;  /* local sesm, kill */
            outFd = shaSesmId2Fd(pSId);
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "KillSesm()->Unknown SesMgr -- Aborted\n")
                    ;
                showInfo(sbOutMsgBuf);
                return;
            }
            putShaTerminateHandler(outFd);
        }
        cmFlush(pHostMainKern->fdSocket);
            return;
    }

    /*
     * Function
     */
    int
    endSystemExportOprn(pSId)
        shastraId      *pSId;
    {
        if (debug) {
            outputId(stderr, pSId);
        }
        checkConn();
        sendReqString(REQ_END_SYSTEM, NULL);
        ShastraIdOut(pHostMainKern->fdSocket, pSId);
        cmFlush(pHostMainKern->fdSocket);
            return(0);
    }
    /*
     * Function
     */
    int
    connectSystemOprn(iObjIndex)
        int             iObjIndex;
    {
        checkConn();
        sendReqString(REQ_CONNECT_SYSTEM, NULL);
        cmFlush(pHostMainKern->fdSocket);
            return(0);
    }
    /*
     * Function
     */
    int
    getShastraIdOprn(iObjIndex)
        int             iObjIndex;
    {
```

```
    checkConn();
    sendReqString(REQ_GET_SHASTRAID, NULL);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
setShastraIdOprn(i)
    int             i;
{

    checkConn();
    sendReqString(REQ_SET_SHASTRAID, NULL);
    getRegisterInfo(&kernelShastraId);
    ShastraIdOut(pHostMainKern->fdSocket, &kernelShastraId);
    printf("%s\n", pSId2Str(&kernelShastraId, PSIDSHOWALL));
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
getShaKernIdOprn(iObjIndex)
    int             iObjIndex;
{
    checkConn();
    sendReqString(REQ_GET_SHAKERNID, NULL);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}


/*
 * Function
 */
int
setShaKernIdOprn(i)
    int             i;
{

    checkConn();
    sendReqString(REQ_SET_SHAKERNID, NULL);
    ShastraIdOut(pHostMainKern->fdSocket, &kernelShastraId);
    printf("%s\n", pSId2Str(&kernelShastraId, PSIDSHOWALL));
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}
```

```
/*
 * Function
 */
void
getShaKernFrIdOprn(iObjIndex)
    int             iObjIndex;
{
    shastraId      *pSId;

    if (fMainKernel) {
        return;
    }
    pSId = shastraKernIds.shastraIds_val[iObjIndex];
    if (pSId->lIPAddr == kernelIPAddr) {
        /* no need to send request for my own data */
        return;
    }
    checkConn();
    sendReqString(REQ_GET_SHAKERNFRID, (char *) NULL);
    ShastraIdOut(pHostMainKern->fdSocket, pSId);
    printf("%s\n", pSId2Str(pSId, PSIDSHOWALL));
    cmFlush(pHostMainKern->fdSocket);
        return;
}


/*
 * Function
 */
int
setShaKernFrIdOprn(i)
    int             i;
{

    checkConn();
    sendReqString(REQ_SET_SHAKERNFRID, NULL);
    ShastraIdOut(pHostMainKern->fdSocket, &kernelShastraId);
    ShastraIdsOut(pHostMainKern->fdSocket, pShastraFrontIds);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
getShaSesmIdOprn(iObjIndex)
    int             iObjIndex;
{
    checkConn();
    sendReqString(REQ_GET_SHASESMID, NULL);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
```

```
}

/*
 * Function
 */
int
setShaSesmIdExportOprn(pSId)
    shastraId      *pSId;
{
    checkConn();
    sendReqString(REQ_SET_SHASESMID, NULL);
    if (debug) {
        outputId(stderr, pSId);
    }
    ShastraIdOut(pHostMainKern->fdSocket, pSId);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}



/*
 * Function
 */
void
getShaSesmFrIdOprn(iObjIndex)
    int             iObjIndex;
{
    shastraIdTag   *pSIdTag;

    if (fMainKernel) {
        return;
    }
    pSIdTag = & shastraSesmIds.shastraIds_val[iObjIndex]->lSIDTag;
    checkConn();
    sendReqString(REQ_GET_SHASESMFRID, (char *) NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, pSIdTag);
    printf("%s\n", pSIdTag2Str(pSIdTag, 0));
    cmFlush(pHostMainKern->fdSocket);
        return;
}

/*
 * Function
 */
int
setShaSesmFrIdExportOprn(pSIdTag, pSIdTags, pPermTags)
    shastraIdTag   *pSIdTag;
    shastraIdTags  *pSIdTags;
    shastraIdTags  *pPermTags;
{
    checkConn();
    sendReqString(REQ_SET_SHASESMFRID, NULL);
    if (debug) {
        outputIdTag(stderr, pSIdTag);
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }
    ShastraIdTagOut(pHostMainKern->fdSocket, pSIdTag);
    ShastraIdTagsOut(pHostMainKern->fdSocket, pSIdTags);
    ShastraIdTagsOut(pHostMainKern->fdSocket, pPermTags);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
deleteSesMgrExportOprn(pSIdTag)
    shastraIdTag   *pSIdTag;
{
    checkConn();
    sendReqString(REQ_DELETE_SESMGR, NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, pSIdTag);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
collInviteJoinOprn(pSesmSIdTag, pFrontSIdTag, pLeaderSIdTag, pFrontPermTag)
    shastraIdTag   *pSesmSIdTag;
    shastraIdTag   *pFrontSIdTag;
    shastraIdTag   *pLeaderSIdTag;
    shastraIdTag   *pFrontPermTag;
{
    checkConn();
    sendReqString(REQ_COLL_INVITEJOIN, NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, pSesmSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pFrontSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pLeaderSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pFrontPermTag);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}
/*
 * Function
 */
int
collAskJoinOprn(pSesmSIdTag, pFrontSIdTag)
    shastraIdTag   *pSesmSIdTag;
    shastraIdTag   *pFrontSIdTag;
{
    checkConn();
    sendReqString(REQ_COLL_ASKJOIN, NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, pSesmSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pFrontSIdTag);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
int
collTellJoinOprn(pSesmSIdTag, pFrontSIdTag, pFrontPermTag)
    shastraIdTag   *pSesmSIdTag;
    shastraIdTag   *pFrontSIdTag;
    shastraIdTag   *pFrontPermTag;
{
    checkConn();
    sendReqString(REQ_COLL_TELLJOIN, NULL);
    ShastraIdTagOut(pHostMainKern->fdSocket, pSesmSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pFrontSIdTag);
    ShastraIdTagOut(pHostMainKern->fdSocket, pFrontPermTag);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}


/*
 * Function
 */
int
helpOprn(iObjIndex)
    int             iObjIndex;
{
    checkConn();
    sendReqString(REQ_HELP, NULL);
    cmFlush(pHostMainKern->fdSocket);
        return(0);
}

/*
 * Function
 */
void
quitOprn(iObjIndex)
    int             iObjIndex;
{
    if (!fMainKernel && (pHostMainKern->fStatus != shaError)) {
        sendReqString(REQ_QUIT, NULL);
        cmFlush(pHostMainKern->fdSocket);
    }
    mplexUnRegisterChannel(pHostMainKern->fdSocket);
    XtDestroyApplicationContext(shastraAppContext);
```

```
        exit(0);
    }


    /*
     * Function
     */
    int
    selectKernOprn(iObjIndex)
        int             iObjIndex;
    {
        fprintf(stderr, "selectKernOprn() -- selected %d\n", iObjIndex);
            return(0);
    }


    /*
     * Function
     */
    int
    startSystemRespHandler(fd)
        int             fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_SYSTEM);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int
    endSystemRespHandler(fd)
        int             fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_SYSTEM);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int
    connectSystemRespHandler(fd)
        int             fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_CONNECT_SYSTEM);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
```

```
 * Function
 */
int
getShastraIdRespHandler(fd)
    int             fd;
{

    ShastraIdsIn(fd, &shastraSysIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASTRAID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, &shastraSysIds);
    }
    if (rgsbShastraSys != NULL) {
        strListDestroy(rgsbShastraSys);
    }
    rgsbShastraSys = pSIds2StrTab(&shastraSysIds, PSIDSHOWALL);
    chooseOneChangeList(pcoShastraSys, rgsbShastraSys,
                coNoInitialHighlight);
        return(0);
}

/*
 * Function
 */
int
setShastraIdRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASTRAID);
    showInfo(sbOutMsgBuf);
    return 0;
}

/*
 * Function
 */
int
getShaKernIdRespHandler(fd)
    int             fd;
{

    ShastraIdsIn(fd, &shastraKernIds);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIds(stderr, &shastraKernIds);
    }
    if (rgsbShastraKern != NULL) {
        strListDestroy(rgsbShastraKern);
    }
    rgsbShastraKern = pSIds2StrTab(&shastraKernIds, PSIDNMHOST);
    chooseOneChangeList(pcoShastraKern, rgsbShastraKern,
            coNoInitialHighlight);

        adjustKrFrMapSize(shastraKernIds.shastraIds_len);
        /* update map */
        updateKrFrMap(&shastraKernIds);
        /* now MCast it to all fronts */
        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putShaKernIdHandler, NULL);
        }
            return(0);
    }

    /*
     * Function
     */
    int
    setShaKernIdRespHandler(fd)
        int             fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHAKERNID);
        showInfo(sbOutMsgBuf);
        return 0;
    }


    /*
     * Function
     */
    int
    getShaKernFrIdRespHandler(fd)
        int             fd;
    {
        int             krIndex;
        int             myIndex;
        static shastraId inShaId;
        static shastraIds inShaIds;
        shastraIds     *pSIds;

        myIndex = locateKernFronts(&kernelShastraId);
        ShastraIdIn(fd, &inShaId);
        krIndex = locateKernFronts(&inShaId);
        /* vaildity check */
        if (krIndex == -1) {
            krIndex = occupyKrFrFreeSlot(&inShaId); /* put him up */
        }
        if (krIndex == myIndex) {
            ShastraIdsIn(fd, &inShaIds);
            pSIds = getKernFrontSIds(&inShaId);
        } else {
            pSIds = getKernFrontSIds(&inShaId);
            ShastraIdsIn(fd, pSIds);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNFRID);
        showInfo(sbOutMsgBuf);
        if (debug) {
            outputIds(stderr, pSIds);
        }
        /* now MCast it to all fronts */
        {
            int            *pfd;
            int             nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putShaKernFrIdHandler, (char *) &inShaId);
        }
            return(0);
    }

    /*
     * Function
     */
    int
    setShaKernFrIdRespHandler(fd)
        int             fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHAKERNFRID);
        showInfo(sbOutMsgBuf);
        return 0;
    }

    /*
     * Function
     */
    int
    getShaSesmIdRespHandler(fd)
        int             fd;
    {

        ShastraIdsIn(fd, &shastraSesmIds);
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMID);
        showInfo(sbOutMsgBuf);
        if (debug) {
            outputIds(stderr, &shastraSesmIds);
        }
        if (rgsbShastraSesMgr != NULL) {
            strListDestroy(rgsbShastraSesMgr);
        }
        rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds, PSIDNMHOST);
        chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
                    coNoInitialHighlight);

        adjustSmFrMapSize(shastraSesmIds.shastraIds_len);
    /* update map */
    updateSmFrMap(&shastraSesmIds);
    /* now MCast it to all fronts */
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmIdHandler, NULL);
    }
        return(0);
}

/*
 * Function
 */
int
setShaSesmIdRespHandler(fd)
    int             fd;
{
    /* proxy.. done for my sesMgrs */
    /* no action need be taken */
        return(0);
}


/*
 * Function
 */
int
getShaSesmFrIdRespHandler(fd)
    int             fd;
{
    int             smIndex;
    static shastraIdTag inShaIdTag;
    static shastraIdTags inShaIdTags;
    shastraIdTags  *pPermTags;
    shastraIdTags  *pSIdTags;

    ShastraIdTagIn(fd, &inShaIdTag);
    smIndex = locateSesmFronts(&inShaIdTag);
    /* vaildity check */
    if (smIndex == -1) {
        fprintf(stderr, "getShaSesmFrIdRespHandler()->can't locate sesMgr!\
            n");
        ShastraIdTagsIn(fd, &inShaIdTags);
        ShastraIdTagsIn(fd, &inShaIdTags);  /* perms */
        return -1;
    }
    pSIdTags = getSesmFrontSIdTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pSIdTags);
    pPermTags = getSesmFrontPermTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pPermTags);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMFRID);
    showInfo(sbOutMsgBuf);
    if (debug) {
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }
    /* now MCast it to all fronts */
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmFrIdHandler, (char *) &inShaIdTag);
    }
        return(0);
}

/*
 * Function
 */
int
setShaSesmFrIdRespHandler(fd)
    int             fd;
{
    /* proxy.. done for my sesMgrs */
    /* no action need be taken */
        return(0);
}

/*
 * Function
 */
int
helpRespHandler(fd)
    int             fd;
{
    standardHelpRespHandler(fd);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_HELP);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
quitRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_QUIT);
    showInfo(sbOutMsgBuf);
        return(0);
}
```

```
/*
 * Function
 */
int
deleteSesMgrRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_DELETE_SESMGR);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
collTellJoinRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}


/*
 * Function
 */
int
collAskJoinRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
collInviteJoinRespHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
collInviteRespHandler(fd)
    int             fd;
{
    shastraIdTag    sesmSIdTag;
    shastraIdTag    frontSIdTag;
    shastraIdTag    leaderSIdTag;
    shastraIdTag    frontPermTag;
    shastraId      *pSId;
    int             outFd;

    ShastraIdTagIn(fd, &sesmSIdTag);
    ShastraIdTagIn(fd, &frontSIdTag);
    ShastraIdTagIn(fd, &leaderSIdTag);
    ShastraIdTagIn(fd, &frontPermTag);

    pSId = krFrSIdTag2SId(frontSIdTag);

    outFd = shaFrontId2Fd(pSId);
    if (outFd == -1) {
        sprintf(sbOutMsgBuf, "collInviteRespHandler()->Unknown Front --
            Aborted\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }
    putCollInviteJoinHandler(outFd, &sesmSIdTag, &frontSIdTag,
        &leaderSIdTag, &frontPermTag);
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
collAskJnRespHandler(fd)
    int             fd;
{
    shastraIdTag    sesmSIdTag;
    shastraIdTag    frontSIdTag;
    shastraId      *pSId;
    int             outFd;

    ShastraIdTagIn(fd, &sesmSIdTag);
    ShastraIdTagIn(fd, &frontSIdTag);

    /*pSId = krFrSIdTag2SId(frontSIdTag);
    outFd = shaFrontId2Fd(pSId);
        */
       pSId = mapSIdTag2SId(&sesmSIdTag);
       outFd = shaSesmId2Fd(pSId);
    if (outFd == -1) {
        sprintf(sbOutMsgBuf, "collAskJnRespHandler()->Unknown Front --
            Aborted\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }
    putCollAskJoinHandler(outFd, &sesmSIdTag, &frontSIdTag);
    sprintf(sbOutMsgBuf, "Done (in)-- %s\n", REQ_COLL_ASKJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}


/*
 * Function
 */
int
collTellJnRespHandler(fd)
    int             fd;
{
    shastraIdTag    sesmSIdTag;
    shastraIdTag    frontSIdTag;
    shastraIdTag    frontPermTag;
    shastraId      *pSId;
    int             outFd;

    ShastraIdTagIn(fd, &sesmSIdTag);
    ShastraIdTagIn(fd, &frontSIdTag);
    ShastraIdTagIn(fd, &frontPermTag);

    pSId = krFrSIdTag2SId(frontSIdTag);

    outFd = shaFrontId2Fd(pSId);
    if (outFd == -1) {
        sprintf(sbOutMsgBuf, "collTellJnRespHandler()->Unknown Front --
            Aborted\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }

    putCollTellJoinHandler(outFd, &sesmSIdTag, &frontSIdTag, &frontPermTag)
        ;
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}


/*
 * Function
 */
int collInviteMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COLL_INVITEMSG, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int collInviteMsgRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collInviteMsgInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collInviteMsgInHandler()")){
        case route_DEFAULT:
            collInviteMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollInviteMsgHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_INVITEMSG);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int collInvRespMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
        hostData* pHostKr;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        char *sbMsg;
    {
        checkConn();
        sendReqString(REQ_COLL_INVRESPMSG, NULL);
        ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
        sendDataString(sbMsg);
        cmFlush(pHostKr->fdSocket);
            return(0);
    }

    /*
     * Function
     */
    int collInvRespMsgRespHandler(fd)
        int fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVRESPMSG);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int collInvRespMsgInHandler(fd)
        int fd;
    {
        /* receive sesm idtag, display recvd message */
        shastraIdTag    smSIdTag;
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        char *sMsg;
        int outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        sMsg = cmReceiveString(fd);
        /*handle*/
        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collInvRespMsgInHandler()")){
            case route_DEFAULT:
                collInvRespMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCollInvRespMsgHandler(outFd, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_INVRESPMSG);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int collInviteStatusReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
        hostData* pHostKr;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        shaULong lStatus;
    {
        checkConn();
        sendReqString(REQ_COLL_INVITESTATUS, NULL);
        ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
        ShastraULongOut(pHostKr->fdSocket, &lStatus);
        cmFlush(pHostKr->fdSocket);
            return(0);
    }

    /*
     * Function
     */
    int collInviteStatusRespHandler(fd)
        int fd;
    {
```

```
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITESTATUS);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collInviteStatusInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd status */
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    shaULong        lStatus;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    ShastraULongIn(fd, &lStatus);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collInviteStatusInHandler()")){
        case route_DEFAULT:
            collInviteStatusReq(pHostMainKern, &smSIdTag, &toSIdTag,
                &sIdTag, lStatus);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollInviteStatusHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, lStatus);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_INVITESTATUS);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collAskJoinMsgReq(pHostKr, pSmSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COLL_ASKJOINMSG, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int collAskJoinMsgRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOINMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collAskJoinMsgInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    smSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeSesMgrSIdTagToFd(&smSIdTag, &outFd,
            "collAskJoinMsgInHandler()")){
        case route_DEFAULT:
            collAskJoinMsgReq(pHostMainKern, &smSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollAskJoinMsgHandler(outFd, &smSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJOINMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}
```

```
/*
 * Function
 */
int collAskJnRespMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COLL_ASKJNRESPMSG, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int collAskJnRespMsgRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNRESPMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collAskJnRespMsgInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collAskJnRespMsgInHandler()")){
        case route_DEFAULT:
            collAskJnRespMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollAskJnRespMsgHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJNRESPMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collAskJnStatusReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
    hostData* pHostKr;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    shaULong lStatus;
{
    checkConn();
    sendReqString(REQ_COLL_ASKJNSTATUS, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pSmSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    ShastraULongOut(pHostKr->fdSocket, &lStatus);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int collAskJnStatusRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNSTATUS);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int collAskJnStatusInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd status */
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    shaULong        lStatus;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    ShastraULongIn(fd, &lStatus);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collAskJnStatusInHandler()")){
        case route_DEFAULT:
            collAskJnStatusReq(pHostMainKern, &smSIdTag, &toSIdTag,
                &sIdTag, lStatus);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollAskJnStatusHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, lStatus);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJNSTATUS);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgTextReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COMM_MSGTEXT, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
```

```
int commMsgTextRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXT);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgTextInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collMsgTextInHandler()")){
        case route_DEFAULT:
            commMsgTextReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgTextHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGTEXT);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgTextFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COMM_MSGTEXTFILE, NULL);
```

```
        ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
        sendDataString(sbMsg);
        cmFlush(pHostKr->fdSocket);
            return(0);
    }

    /*
     * Function
     */
    int commMsgTextFileRespHandler(fd)
        int fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXTFILE);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int commMsgTextFileInHandler(fd)
        int fd;
    {
        /* receive sesm idtag, display recvd message */
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        char *sMsg;
        int outFd;

        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        sMsg = cmReceiveString(fd);
        /*handle*/
        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collMsgTextFileInHandler()")){
            case route_DEFAULT:
                commMsgTextFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCommMsgTextFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGTEXTFILE);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
```

```
 * Function
 */
int commMsgAudioReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COMM_MSGAUDIO, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int commMsgAudioRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIO);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgAudioInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collMsgAudioInHandler()")){
        case route_DEFAULT:
            commMsgAudioReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgAudioHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGAUDIO);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgAudioFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COMM_MSGAUDIOFILE, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int commMsgAudioFileRespHandler(fd)
    int fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIOFILE);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgAudioFileInHandler(fd)
    int fd;
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
```

```
        /*handle*/
        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collMsgAudioFileInHandler()")){
            case route_DEFAULT:
                commMsgAudioFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCommMsgAudioFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGAUDIOFILE);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int commMsgVideoReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
        hostData* pHostKr;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        char *sbMsg;
    {
        checkConn();
        sendReqString(REQ_COMM_MSGVIDEO, NULL);
        ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
        ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
        sendDataString(sbMsg);
        cmFlush(pHostKr->fdSocket);
            return(0);
    }

    /*
     * Function
     */
    int commMsgVideoRespHandler(fd)
        int fd;
    {
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEO);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int commMsgVideoInHandler(fd)
        int fd;
```

```
{
    /* receive sesm idtag, display recvd message */
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    /*handle*/
    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collMsgVideoInHandler()")){
        case route_DEFAULT:
            commMsgVideoReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgVideoHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGVIDEO);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int commMsgVideoFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
    hostData* pHostKr;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    checkConn();
    sendReqString(REQ_COMM_MSGVIDEOFILE, NULL);
    ShastraIdTagOut(pHostKr->fdSocket, pToSIdTag);
    ShastraIdTagOut(pHostKr->fdSocket, pSIdTag);
    sendDataString(sbMsg);
    cmFlush(pHostKr->fdSocket);
        return(0);
}

/*
 * Function
 */
int commMsgVideoFileRespHandler(fd)
    int fd;
{
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEOFILE);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int commMsgVideoFileInHandler(fd)
        int fd;
    {
        /* receive sesm idtag, display recvd message */
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        char *sMsg;
        int outFd;

        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        sMsg = cmReceiveString(fd);
        /*handle*/
        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collMsgVideoFileInHandler()")){
            case route_DEFAULT:
                commMsgVideoFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCommMsgVideoFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGVIDEOFILE);
        showInfo(sbOutMsgBuf);
            return(0);
    }
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>
#include <sys/wait.h>
#include <netdb.h>
#include <stdlib.h>
#ifdef SHASTRA4SUN5
#include <unistd.h>
char *strdup(char *);
#endif

#include <shastra/shastra.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/chooseMany.h>
#include <shastra/uitools/callbackArg.h>
#include <shastra/uitools/strListUtilities.h>
#include <shastra/ui/general.h>

#include <shastra/kernel/kernel.h>
#include <shastra/kernel/kernelMainCB.h>
#include <shastra/kernel/kernel_server.h>
#include <shastra/kernel/kernel_client.h>
#include <shastra/kernel/kernelState.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>

#include <shastra/datacomm/shastraDataH.h>
```

```
#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFrontsP.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

int closedChannelCleanUpHandler(Prot1(int ));
int putCollTellJoinHandler(Prot4( int, shastraIdTag *, shastraIdTag *,
    shastraIdTag *));

int putCollAskJoinHandler(Prot3(int , shastraIdTag   *, shastraIdTag   *));
int quitFrontCleanUpHandler(Prot1(int));
int quitSesMgrCleanUpHandler(Prot1(int));
int commMsgTextFileReq(Prot4( hostData* , shastraIdTag *, shastraIdTag *,
    char *sbMsg));
void deleteSesMgrExportOprn(Prot1( shastraIdTag   *));
int quitKernelCleanUpHandler(Prot1(int));
extern int shaSesmId2Fd();
extern int cmAckError();




#define DEBUGnn
extern int      debug;
char           *shaAppSesmMap[][2] =  SHA_APPSESM_MAP ;
#define SHA_APPSESM_MAP_SIZE (sizeof(shaAppSesmMap)/(2*sizeof(char*)))

#define putStringOnChannel(filedesc, reqstr, funcstr)       \
    if (cmSendString(filedesc, reqstr) == -1) {     \
        fprintf(stderr, "%s : Error Sending to %d\n", funcstr, filedesc);
                \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define sendDataString(fd, s)       \
    if(cmSendString(fd, s) == -1){  \
        fprintf(stderr,"Error in Sending Operation Data\n");    \
        closedChannelCleanUpHandler(pHostMainKern->fdSocket);       \
        return(0);
                \
    }

#define ShastraIdIn(filedesc, pShaId)           \
    if(shastraIdIn(filedesc, pShaId) == -1){    \
        fprintf(stderr, "Error Receiving SID from %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }
```

```
#define ShastraIdOut(filedesc, pShaId)          \
    if(shastraIdOut(filedesc, pShaId) == -1){   \
        fprintf(stderr, "Error Sending SID to %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraIdsIn(filedesc, pShaIds)         \
    if(shastraIdsIn(filedesc, pShaIds) == -1){  \
        fprintf(stderr, "Error Receiving SIDs from %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraIdsOut(filedesc, pShaIds)            \
    if(shastraIdsOut(filedesc, pShaIds) == -1){ \
        fprintf(stderr, "Error Sending SIDs to %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }


#define ShastraIdTagIn(filedesc, pShaIdTag)         \
    if(shastraIdTagIn(filedesc, pShaIdTag) == -1){  \
        fprintf(stderr, "Error Receiving SID from %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraIdTagOut(filedesc, pShaIdTag)            \
    if(shastraIdTagOut(filedesc, pShaIdTag) == -1){ \
        fprintf(stderr, "Error Sending SID to %d\n", filedesc); \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraIdTagsIn(filedesc, pShaIdTags)           \
    if(shastraIdTagsIn(filedesc, pShaIdTags) == -1){    \
        fprintf(stderr, "Error Receiving SIDs from %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraIdTagsOut(filedesc, pShaIdTags)          \
    if(shastraIdTagsOut(filedesc, pShaIdTags) == -1){   \
        fprintf(stderr, "Error Sending SIDs to %d\n", filedesc);    \
        closedChannelCleanUpHandler(filedesc);                  \
        return(0);                                          \
    }

#define ShastraULongIn(filedesc, pShaIdTag)         \
    if(shaULongIn(filedesc, pShaIdTag) == -1){  \
        fprintf(stderr, "Error Receiving ULong from %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return(0);                                              \
        }

    #define ShastraULongOut(filedesc, pShaIdTag)            \
        if(shaULongOut(filedesc, pShaIdTag) == -1){ \
            fprintf(stderr, "Error Sending ULong to %d\n", filedesc);   \
            closedChannelCleanUpHandler(filedesc);                  \
            return(0);                                              \
        }


    shaRouteMode
    routeFrontSIdTagToFd(pSIdTag, pFd, nmFunc)
        shastraIdTag *pSIdTag;
        int *pFd;
        char *nmFunc;
    {
        shastraId *pSId;
        int outFd = -1;
        shaRouteMode retVal = route_ERROR;

        pSId = krFrSIdTag2SId(*pSIdTag);
        if (pSId == NULL) {
            sprintf(sbOutMsgBuf, "%s->Unknown IDTag -- Aborted\n", nmFunc);
            showInfo(sbOutMsgBuf);
            return(retVal);
        }
        if (pSId->lIPAddr != kernelShastraId.lIPAddr) {
            if (fMainKernel) {
                outFd = shaKernId2Fd(pSId);
                if (outFd == -1) {
                    sprintf(sbOutMsgBuf, "%s->Unknown Kernel -- Aborted\n",
                        nmFunc);
                    showInfo(sbOutMsgBuf);
                    return retVal;
                }
                else{
                    retVal = route_KERNEL;
                }
            } else {
                retVal = route_DEFAULT;
            }
        } else {
            outFd = shaFrontId2Fd(pSId);
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "%s->Unknown Front -- Aborted\n", nmFunc);
                showInfo(sbOutMsgBuf);
                return retVal;
            }
            else{
                retVal = route_FRONT;
            }
        }
        *pFd = outFd;
        return retVal;
    }

    shaRouteMode
    routeSesMgrSIdTagToFd(pSIdTag, pFd, nmFunc)
        shastraIdTag *pSIdTag;
        int *pFd;
        char *nmFunc;
    {
        shastraId *pSId;
        int outFd = -1;
        shaRouteMode retVal = route_ERROR;

        pSId = getSIdByTagInSIds(pSIdTag, &shastraSesmIds);
        if (pSId == NULL) {
            sprintf(sbOutMsgBuf, "%s->Unknown Sesm IDTag -- Aborted\n", nmFunc)
                ;
            showInfo(sbOutMsgBuf);
            return retVal;
        }
        if (pSId->lIPAddr != kernelShastraId.lIPAddr) {
            if (fMainKernel) {
                outFd = shaKernId2Fd(pSId);
                if (outFd == -1) {
                    sprintf(sbOutMsgBuf, "%s->Unknown Kernel -- Aborted\n",
                        nmFunc);
                    showInfo(sbOutMsgBuf);
                    return retVal;
                }
                else{
                    retVal = route_KERNEL;
                }
            } else {
                retVal = route_DEFAULT;
            }
        } else {
            outFd = shaSesmId2Fd(pSId);
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "%s->Unknown SesMgr -- Aborted\n", nmFunc)
                    ;
                showInfo(sbOutMsgBuf);
                return(0);
            }
            else{
                retVal = route_SESMGR;
            }
        }
        *pFd = outFd;
        return retVal;
    }
```

```
shaRouteMode
routeKernelSIdTagToFd(pSIdTag, pFd, nmFunc)
    shastraIdTag *pSIdTag;
    int *pFd;
    char *nmFunc;
{
    shastraId *pSId;
    int outFd = -1;
    shaRouteMode retVal = route_ERROR;

    pSId = getSIdByTagInSIds(pSIdTag, &shastraKernIds);
    if (pSId == NULL) {
        sprintf(sbOutMsgBuf, "%s->Unknown Kernel IDTag -- Aborted\n",
            nmFunc);
        showInfo(sbOutMsgBuf);
        return retVal;
    }
    if (pSId->lIPAddr != kernelShastraId.lIPAddr) {
        if (fMainKernel) {
            outFd = shaKernId2Fd(pSId);
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "%s->Unknown Kernel -- Aborted\n",
                    nmFunc);
                showInfo(sbOutMsgBuf);
                return retVal;
            }
            else{
                retVal = route_KERNEL;
            }
        } else {
            retVal = route_DEFAULT;
        }
    } else {
        retVal = route_SELF;
    }
    *pFd = outFd;
    return retVal;
}

/*
 * Function
 */
int
startSystemHandler(fd)
    int             fd;
{
    static shastraId createSId;
    static char     rshCmdBuf[256];
    char           *shastraArgv[6];
    int             retVal;
    int             outFd;
    int             krIndex;
    shastraIdTag   *pSIdTag;
    shastraId       *pSIdTmp;

    ShastraIdIn(fd, &createSId);
    if (debug) {
        outputId(stderr, &createSId);
    }
    if (!strcmp(createSId.nmHost, kernelHostName)) {
        if (!strcmp(createSId.nmUser, kernelShastraId.nmUser)) {
            sprintf(rshCmdBuf, "%s", createSId.nmApplicn);
            shastraArgv[0] = rshCmdBuf;
            shastraArgv[1] = "-display";
            shastraArgv[2] = createSId.nmDisplay;
            shastraArgv[3] = "-passwd";
            shastraArgv[4] = createSId.nmPasswd;
            shastraArgv[5] = NULL;
#ifdef SHASTRA4SUN4
            if (vfork() == 0)
#else               /* SHASTRA4SUN4 */
            if (fork() == 0)
#endif              /* SHASTRA4SUN4 */
            {
                execv(shastraArgv[0], shastraArgv);
                return(0);
            } else {
                wait3(NULL, WNOHANG, NULL);
            }
        } else {
            fprintf(stderr, "startSystemHandler()->can't start system for
                other users!\n");
        }
    }
    else if ((krIndex = locateByNameKernFronts(&createSId)) != -1) {
        if (fMainKernel) {

            pSIdTag = KernFrontSIdTag(krIndex);
            pSIdTmp = getSIdByTagInSIds(pSIdTag, &shastraKernIds);
            outFd = shaKernId2Fd(pSIdTmp);
            if (outFd == -1) {
                sprintf(sbOutMsgBuf, "Create()->Unknown Kernel -- Aborted\
                    n");
                showInfo(sbOutMsgBuf);
                cmAckError(fd);
                cmFlush(fd);

                return(0);
            }
            putShaStartSysHandler(outFd, &createSId);
        } else {
            startSystemExportOprn(&createSId);
        }
    } else {
        if (!strcmp(createSId.nmUser, kernelShastraId.nmUser)) {
            sprintf(rshCmdBuf, "echo \"cd shastra;\nexec %s -display %s -
                passwd %s </dev/null >/dev/null 2>&1 &\" | rsh %s /bin/sh\
                n",
                createSId.nmApplicn, createSId.nmDisplay,
                createSId.nmPasswd, createSId.nmHost);
            retVal = system(rshCmdBuf);
            fprintf(stdout, "%s\nretVal = %d\n", rshCmdBuf, retVal);
        } else {
            fprintf(stderr, "startSystemHandler()->can't start system for
                other users!\n");
        }
    }
    if (fd != mainKernClntSocket) {
        cmAckOk(fd);
        cmFlush(fd);

    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_START_SYSTEM);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
endSystemHandler(fd)
    int             fd;
{
    static shastraId killSId;
    int             outFd;
    int             krIndex;

    ShastraIdIn(fd, &killSId);
    if (debug) {
        outputId(stderr, &killSId);
    }
    if (!strcmp(killSId.nmHost, kernelHostName)) {
        outFd = shaFrontId2Fd(&killSId);
        if (outFd == -1) {
            if (killSId.lSIDTag == kernelShastraId.lSIDTag) {
                terminateHandler(0);
            } else {
                outFd = shaSesmId2Fd(&killSId);
                if (outFd == -1) {
                    cmAckError(fd);
                    cmFlush(fd);
    sprintf(sbOutMsgBuf, "endSystemHandler() -- unknown system\n");
    showInfo(sbOutMsgBuf);
                    return(0);
                } else {
                    putShaTerminateHandler(outFd);
                }
            }
        } else {
            putShaTerminateHandler(outFd);
        }
    }
    else if ((krIndex = locateByNameKernFronts(&killSId)) != -1) {
        if (fMainKernel) {
            outFd = shaKernId2Fd(&killSId);
            if (outFd == -1) {
    sprintf(sbOutMsgBuf, "KillHandler()->Unknown Kernel -- Aborted\n");
    showInfo(sbOutMsgBuf);
                cmAckError(fd);
                cmFlush(fd);

                return(0);
            }
            putShaEndSysHandler(outFd, &killSId);
        } else {
            endSystemExportOprn(&killSId);
        }
    } else {
        cmAckError(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "endSystemHandler() -- unknown host\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }
    if (fd != mainKernClntSocket) {
        if (fd != outFd) {
            cmAckOk(fd);
            cmFlush(fd);
        }
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_END_SYSTEM);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
connectSystemHandler(fd)
    int             fd;
{
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_CONNECT_SYSTEM);
    showInfo(sbOutMsgBuf);
    cmAckOk(fd);
    cmFlush(fd);
        return(0);
}

/*
```

```
 * Function
 */
int
getShastraIdHandler(fd)
    int             fd;
{
    cmAckOk(fd);
    ShastraIdsOut(fd, pShastraFrontIds);
    cmFlush(fd);

    if (debug) {
        outputIds(stderr, pShastraFrontIds);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASTRAID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
setShastraIdHandler(fd)
    int             fd;
{
    shastraId      *pSId;

    pSId = &localShaIdIn[fd];
    shaKernFlags[fd] = SHAFRONT;
    ShastraIdIn(fd, pSId);
    if (debug) {
        outputId(stderr, pSId);
    }
    updateShaFrontIds(pShastraFrontIds);

    if (rgsbShastraFront != NULL) {
        strListDestroy(rgsbShastraFront);
    }
    rgsbShastraFront = pSIds2StrTab(pShastraFrontIds, PSIDNMHOST |
        PSIDNMAPPL);
    chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                coNoInitialHighlight);

    cmAckOk(fd);
    putShaStateHandler(fd);
    if (!fMainKernel)
        {
        setShaKernFrIdOprn(0);
    }
        {
        int             *pfd;
        int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaKernFrIdHandler,
                    (char *) &kernelShastraId);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASTRAID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
getShaKernIdHandler(fd)
    int             fd;
{

    cmAckOk(fd);
    ShastraIdsOut(fd, &shastraKernIds);
    cmFlush(fd);

    if (debug) {
        outputIds(stderr, &shastraKernIds);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
setShaKernIdHandler(fd)
    int             fd;
{
    shastraId      *pSId;
    int             krIndex;

    pSId = &localShaIdIn[fd];
    shaKernFlags[fd] = SHAKERNEL;
    ShastraIdIn(fd, pSId);
    if (debug) {
        outputId(stderr, pSId);
    }
    if (!fMainKernel) {
        cmAckError(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "setShaKernIdHandler() -- Not Authorized\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }
    updateShaKernIds();
    krIndex = locateKernFronts(pSId);
    if (krIndex == -1) {
        krIndex = occupyKrFrFreeSlot(pSId);
    } else {
        fprintf(stderr, "setShaKernIdHandler()-- already in %d\n", krIndex)
            ;
    }

    if (rgsbShastraKern != NULL) {
        strListDestroy(rgsbShastraKern);
    }
    rgsbShastraKern = pSIds2StrTab(&shastraKernIds, PSIDNMHOST);
    chooseOneChangeList(pcoShastraKern, rgsbShastraKern,
                coNoInitialHighlight);

    cmAckOk(fd);
    putShaStateHandler(fd);
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaKernIdHandler, NULL);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHAKERNID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
getShaKernFrIdHandler(fd)
    int             fd;
{
    static shastraId inShaId;
    shastraIds     *pSIds;
    int             kernFd = -1;

    ShastraIdIn(fd, &inShaId);
    kernFd = locateKernFronts(&inShaId);
    if (kernFd == -1) {
        cmAckError(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "getShaKernFrIdHandler() -- unknown kernel\n")
            ;
        showInfo(sbOutMsgBuf);
        return(0);
    }
    pSIds = getKernFrontSIds(&inShaId);
    cmAckOk(fd);
    ShastraIdOut(fd, &inShaId);
    ShastraIdsOut(fd, pSIds);
    cmFlush(fd);

    if (debug) {
        outputId(stderr, &inShaId);
        outputIds(stderr, pSIds);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNFRID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
setShaKernFrIdHandler(fd)
    int             fd;
{
    shastraIds     *pSIds;
    static shastraId inShaId;
    static shastraIds inShaIds;
    int             krIndex;
    int             myIndex;

    myIndex = locateKernFronts(&kernelShastraId);
    ShastraIdIn(fd, &inShaId);
    krIndex = locateKernFronts(&inShaId);
    if (krIndex == -1) {
        fprintf(stderr, "setShaKernFrIdHandler()-> unlocated kernel!\n");
        ShastraIdsIn(fd, &inShaIds);
        cmAckError(fd);
        cmFlush(fd);

        return -1;
    }
    if (krIndex == myIndex) {
        ShastraIdsIn(fd, &inShaIds);
        cmAckError(fd);
        cmFlush(fd);

        return 0;
    }
    pSIds = getKernFrontSIds(&inShaId);
    ShastraIdsIn(fd, pSIds);
    if (debug) {
        outputId(stderr, &inShaId);
        outputIds(stderr, pSIds);
    }
    if (!fMainKernel) {
            cmAckError(fd);
            cmFlush(fd);

            sprintf(sbOutMsgBuf, "setShaKernFrIdHandler() -- Not Authorized\n")
                ;
            showInfo(sbOutMsgBuf);
            return -1;
        }
        cmAckOk(fd);
        cmFlush(fd);

        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putShaKernFrIdHandler, (char *) &inShaId);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHAKERNFRID);
        showInfo(sbOutMsgBuf);
            return(0);
    }


    /*
     * Function
     */
    int
    getShaSesmIdHandler(fd)
        int             fd;
    {

        cmAckOk(fd);
        ShastraIdsOut(fd, &shastraSesmIds);
        cmFlush(fd);

        if (debug) {
            outputIds(stderr, &shastraSesmIds);
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMID);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int
    setShaSesmIdHandler(fd)
        int             fd;
    {
        static shastraId inShaId;
        shastraId      *pSId;
    int             iLoc;

    if (shaKernFlags[fd] == SHAKERNEL) {
        pSId = &inShaId;
    } else {
        pSId = &localShaIdIn[fd];
        shaKernFlags[fd] = SHASESMGR;
    }
    ShastraIdIn(fd, pSId);
    if (debug) {
        outputId(stderr, pSId);
    }

    iLoc = getSIdIndexInSIds(pSId, &shastraSesmIds);
    if (iLoc == -1) {
        addSId2SIds(pSId, &shastraSesmIds);
        if (occupySmFrFreeSlot(& pSId->lSIDTag) == -1) {
            fprintf(stderr, "setShaSesmIdHandler()->couldn't occupy slot!\
                n");
        }
    } else {
        fprintf(stderr, "setShaSesmIdHandler()->already occupied slot!\n");
    }

    if (rgsbShastraSesMgr != NULL) {
        strListDestroy(rgsbShastraSesMgr);
    }
    rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds,
                     PSIDNMHOST | PSIDNMAPPL);
    chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
                coNoInitialHighlight);

    if (shaKernFlags[fd] == SHASESMGR) {
        cmAckOk(fd);
        putShaStateHandler(fd);
    }
    if (!fMainKernel) {
        setShaSesmIdExportOprn(pSId);
    } {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmIdHandler, (char *) pSId);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASESMID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
```

```
int
getShaSesmFrIdHandler(fd)
    int             fd;
{
    static shastraIdTag inShaIdTag;
    shastraIdTags  *pSIdTags;
    shastraIdTags  *pPermTags;
    int             smIndex = -1;

    ShastraIdTagIn(fd, &inShaIdTag);
    smIndex = locateSesmFronts(&inShaIdTag);
    if (smIndex == -1) {
        cmAckError(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "getShaSesmFrIdHandler() -- unknown sesMgr\n")
            ;
        showInfo(sbOutMsgBuf);
        return(0);
    }
    ShastraIdTagOut(fd, &inShaIdTag);
    pSIdTags = getSesmFrontSIdTags(&inShaIdTag);
    pPermTags = getSesmFrontPermTags(&inShaIdTag);

    cmAckOk(fd);
    ShastraIdTagsOut(fd, pSIdTags);
    ShastraIdTagsOut(fd, pPermTags);
    cmFlush(fd);

    if (debug) {
        outputIdTag(stderr, &inShaIdTag);
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_GET_SHASESMFRID);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int
setShaSesmFrIdHandler(fd)
    int             fd;
{
    shastraIdTags  *pSIdTags;
    shastraIdTags  *pPermTags;
    static shastraIdTag inShaIdTag;
    static shastraIdTags inShaIdTags;
    static shastraIdTags inShaPermTags;
    int             smIndex;
    ShastraIdTagIn(fd, &inShaIdTag);
    smIndex = locateSesmFronts(&inShaIdTag);
    if (smIndex == -1) {
        fprintf(stderr, "setShaSesmFrIdHandler()-> unlocated sesMgr!\n");
        ShastraIdTagsIn(fd, &inShaIdTags);
        ShastraIdTagsIn(fd, &inShaPermTags);
        cmAckError(fd);
        cmFlush(fd);

        return(0);
    }
    pSIdTags = getSesmFrontSIdTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pSIdTags);
    pPermTags = getSesmFrontPermTags(&inShaIdTag);
    ShastraIdTagsIn(fd, pPermTags);
    if (debug) {
        outputIdTag(stderr, &inShaIdTag);
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }

    cmAckOk(fd);
    cmFlush(fd);

    if (!fMainKernel) {
        setShaSesmFrIdExportOprn(&inShaIdTag, pSIdTags, pPermTags);
    } {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmFrIdHandler, (char *) &inShaIdTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_SET_SHASESMFRID);
    showInfo(sbOutMsgBuf);
        return(0);
}

int
helpHandler(fd)
    int             fd;
{
    int             i;
    char            buf[512];

    cmAckOk(fd);
    sprintf(buf, "%d\n", serverNCmds);
    putStringOnChannel(fd, buf, "helpHandler()");
    for (i = 0; i < serverNCmds; i++) {
        sprintf(buf, "%s -- %s\n", serverCommandTab[i].command,
            serverCommandTab[i].helpmsg);
        putStringOnChannel(fd, buf, "helpHandler()");
    }
    cmFlush(fd);

    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_HELP);
    showInfo(sbOutMsgBuf);
        return(0);
}

int
quitHandler(fd)
    int             fd;
{
    int             fKern;

    fKern = shaKernFlags[fd];

    switch (fKern) {
    case SHAKERNEL:
        quitKernelCleanUpHandler(fd);
        break;
    case SHASESMGR:
        quitSesMgrCleanUpHandler(fd);
        break;
    case SHAFRONT:
        quitFrontCleanUpHandler(fd);
        break;
    default:
        fprintf(stderr, "quitHandler()-> shouldn't happen!\n");
        break;
    }
        return(0);
}


int
quitKernelCleanUpHandler(fd)
    int             fd;
{
    mplexUnRegisterChannel(fd);
    deleteShaIdFromTab(fd, pShastraFrontIds);
    if (rgsbShastraKern != NULL) {
        strListDestroy(rgsbShastraKern);
    }
    rgsbShastraKern = pSIds2StrTab(&shastraKernIds, PSIDNMHOST);
    chooseOneChangeList(pcoShastraKern, rgsbShastraKern,
                coNoInitialHighlight);

    if (!fMainKernel) {
        fprintf(stderr, "quitKernelHandler()-> shouldn't happen!\n");
    }
    localShaIdIn[fd].lSIDTag = 0;
    {
        int             *pfd;
        int              nfd;
        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaKernIdHandler, NULL);
    }
    sprintf(sbOutMsgBuf, "Done (Kernel)-- %s\n", REQ_QUIT);
    showInfo(sbOutMsgBuf);
        return(0);
}

int
quitSesMgrCleanUpHandler(fd)
    int             fd;
{
    mplexUnRegisterChannel(fd);
    shaKernFlags[fd] = 0;
    deleteSIdFromSIds(&localShaIdIn[fd], &shastraSesmIds);
    freeSmFrSlot(& localShaIdIn[fd].lSIDTag);

    if (rgsbShastraSesMgr != NULL) {
        strListDestroy(rgsbShastraSesMgr);
    }
    rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds,
                    PSIDNMHOST | PSIDNMAPPL);
    chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
                coNoInitialHighlight);

    if (!fMainKernel) {
        deleteSesMgrExportOprn( & localShaIdIn[fd].lSIDTag);
    }
    localShaIdIn[fd].lSIDTag = 0;
    {
        int             *pfd;
        int              nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaSesmIdHandler, NULL);
    }
    sprintf(sbOutMsgBuf, "Done (SesMgr)-- %s\n", REQ_QUIT);
    showInfo(sbOutMsgBuf);
        return(0);
}

int
quitFrontCleanUpHandler(fd)
    int             fd;
{
    mplexUnRegisterChannel(fd);
    deleteShaIdFromTab(fd, pShastraFrontIds);
    if (rgsbShastraFront != NULL) {
        strListDestroy(rgsbShastraFront);
    }
    rgsbShastraFront = pSIds2StrTab(pShastraFrontIds,
                    PSIDNMHOST | PSIDNMAPPL);
    chooseOneChangeList(pcoShastraFront, rgsbShastraFront,
                coNoInitialHighlight);

    if (!fMainKernel) {
        setShaKernFrIdOprn(0);
    }
    localShaIdIn[fd].lSIDTag = 0;
    {
        int            *pfd;
        int             nfd;

        getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
        cmMultiCast(pfd, nfd, putShaKernFrIdHandler,
                (char *) &kernelShastraId);
    }
    sprintf(sbOutMsgBuf, "Done (Front)-- %s\n", REQ_QUIT);
    showInfo(sbOutMsgBuf);
        return(0);
}

int
collInitiateHandler(fd)
    int             fd;
{
    char          **shastraArgv;
    static shastraIdTags sIdTags;
    char            sbBuf[32];
    int             i,n;
    shastraId      *pSId;
    unsigned long   perms, lIdTag;
    char *sName;

    ShastraIdTagsIn(fd, &sIdTags);
    ShastraULongIn(fd, &perms);
    ShastraULongIn(fd, &lIdTag);
    if (debug) {
        outputIdTags(stderr, &sIdTags);
    }

    pSId = krFrSIdTag2SId(sIdTags.shastraIdTags_val[0]);
    if (pSId == NULL) {
        fprintf(stderr, "collInitiateHandler()->type unknown.. aborting\n")
            ;
        cmAckError(fd);
        cmFlush(fd);

        return(0);
    }
    shastraArgv = (char **) malloc(sizeof(char *) *
        (sIdTags.shastraIdTags_len + 16));

  sName = resolveNameFrom2Bases(pKernelAppData->sDirBase,
        pKernelAppData->sDirBin, pKernelAppData->sLocStart);
    n = 0;
    shastraArgv[n++] = strdup(sName);
    for (i = 0; i < SHA_APPSESM_MAP_SIZE; i++) {
        if (!strcmp(pSId->nmApplicn, shaAppSesmMap[i][0])) {
            shastraArgv[n++] = strdup(shaAppSesmMap[i][1]);
            break;
        }
    }
    if (i == SHA_APPSESM_MAP_SIZE) {
        fprintf(stderr, "collInitiateHandler()->No SesMgr.. aborting\n");
        cmAckError(fd);
        cmFlush(fd);

        return(0);
    }
    shastraArgv[n++] = strdup("-display");
    shastraArgv[n++] = strdup(kernelDispName);
    shastraArgv[n++] = strdup("-passwd");
    shastraArgv[n++] = strdup(kernelPasswd);
    shastraArgv[n++] = strdup("-perms");
    sprintf(sbBuf, "%lu", perms);
    shastraArgv[n++] = strdup(sbBuf);
    shastraArgv[n++] = strdup("-idtag");
    sprintf(sbBuf, "%lu", lIdTag);
    shastraArgv[n++] = strdup(sbBuf);
    shastraArgv[n++] = strdup("-tags");
    for (i = 0; i < sIdTags.shastraIdTags_len; i++) {
        sprintf(sbBuf, "%lu", sIdTags.shastraIdTags_val[i]);
        shastraArgv[n++] = strdup(sbBuf);
    }
    shastraArgv[n++] = NULL;
#ifdef SHASTRA4SUN4
    if (vfork() == 0)
#else               /* SHASTRA4SUN4 */
    if (fork() == 0)
#endif              /* SHASTRA4SUN4 */
    {
        execv(shastraArgv[0], shastraArgv);
        return(0);
    } else {
        strListDestroy(shastraArgv);
        wait3(NULL, WNOHANG, NULL);
        cmAckOk(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INITIATE);
        showInfo(sbOutMsgBuf);
    }
        return(0);
}

int
```

```
collAutoInitiateHandler(fd)
    int             fd;
{
    char          **shastraArgv;
    static shastraIdTags sIdTags;
    char            sbBuf[32];
    int             i, n;
    shastraId      *pSId;
    unsigned long   perms, lIdTag;
    char *sName;

        n = 0;
    ShastraIdTagsIn(fd, &sIdTags);
    ShastraULongIn(fd, &perms);
    ShastraULongIn(fd, &lIdTag);
    if (debug) {
        outputIdTags(stderr, &sIdTags);
    }

    pSId = krFrSIdTag2SId(sIdTags.shastraIdTags_val[0]);
    if (pSId == NULL) {
        fprintf(stderr, "collInitiateHandler()->type unknown.. aborting\n")
            ;
        cmAckError(fd);
        cmFlush(fd);

        return(0);
    }
    shastraArgv = (char **) malloc(sizeof(char *) *
        (sIdTags.shastraIdTags_len + 13));

  sName = resolveNameFrom2Bases(pKernelAppData->sDirBase,
        pKernelAppData->sDirBin, pKernelAppData->sLocStart);
    shastraArgv[n++] = strdup(sName);

    for (i = 0; i < SHA_APPSESM_MAP_SIZE; i++) {
        if (!strcmp(pSId->nmApplicn, shaAppSesmMap[i][0])) {
            shastraArgv[n++] = strdup(shaAppSesmMap[i][1]);
            break;
        }
    }
    if (i == SHA_APPSESM_MAP_SIZE) {
        fprintf(stderr, "collInitiateHandler()->No SesMgr.. aborting\n");
        cmAckError(fd);
        cmFlush(fd);

        return(0);
    }
    shastraArgv[n++] = strdup("-display");
    shastraArgv[n++] = strdup(kernelDispName);
    shastraArgv[n++] = strdup("-passwd");
    shastraArgv[n++] = strdup(kernelPasswd);
    shastraArgv[n++] = strdup("-auto");
    shastraArgv[n++] = strdup("-perms");
    sprintf(sbBuf, "%lu", perms);
    shastraArgv[n++] = strdup(sbBuf);
    shastraArgv[n++] = strdup("-idtag");
    sprintf(sbBuf, "%lu", lIdTag);
    shastraArgv[n++] = strdup(sbBuf);
    shastraArgv[n++] = strdup("-tags");
    for (i = 0; i < sIdTags.shastraIdTags_len; i++) {
        sprintf(sbBuf, "%lu", sIdTags.shastraIdTags_val[i]);
        shastraArgv[n++] = strdup(sbBuf);
    }
    shastraArgv[n++] = NULL;
#ifdef SHASTRA4SUN4
    if (vfork() == 0)
#else               /* SHASTRA4SUN4 */
    if (fork() == 0)
#endif              /* SHASTRA4SUN4 */
    {
        execv(shastraArgv[0], shastraArgv);
        return(0);
    } else {
        strListDestroy(shastraArgv);
        wait3(NULL, WNOHANG, NULL);
        cmAckOk(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INITIATE);
        showInfo(sbOutMsgBuf);
    }
        return(0);
}

int
deleteSesMgrHandler(fd)
    int             fd;
{
    static shastraIdTag sIdTag;
    int             iSm;

    if (!fMainKernel) {
        cmAckError(fd);
        cmFlush(fd);

        fprintf(stderr, "deleteSesMgrHandler()-> shouldn't happen\n");
        return(0);
    }
    ShastraIdTagIn(fd, &sIdTag);
    iSm = getSIdTagIndexInSIds(&sIdTag, &shastraSesmIds);
    if (iSm == -1) {
        cmAckError(fd);
        cmFlush(fd);

        sprintf(sbOutMsgBuf, "%s.. no such sesMgr\n", REQ_DELETE_SESMGR);
```

```
            showInfo(sbOutMsgBuf);
            return(0);
        }
        cmAckOk(fd);
        cmFlush(fd);

        deleteSIdFromSIds(shastraSesmIds.shastraIds_val[iSm], &shastraSesmIds);
        freeSmFrSlot(&sIdTag);
        if (rgsbShastraSesMgr != NULL) {
            strListDestroy(rgsbShastraSesMgr);
        }
        rgsbShastraSesMgr = pSIds2StrTab(&shastraSesmIds,
                         PSIDNMHOST | PSIDNMAPPL);
        chooseOneChangeList(pcoShastraSesMgr, rgsbShastraSesMgr,
                    coNoInitialHighlight);

        {
            int             *pfd;
            int              nfd;

            getKrFDsMCast(fd, &pfd, &nfd, shastraServiceSocket);
            cmMultiCast(pfd, nfd, putShaSesmIdHandler,
                    (char *) &kernelShastraId);
        }

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_DELETE_SESMGR);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    int
    terminateHandler(fd)
        int             fd;
    {

        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_TERMINATE);
        showInfo(sbOutMsgBuf);
        quitOprn(0);
            return(0);
    }
    int
    collInviteJoinHandler(fd)
        int             fd;
    {
        shastraIdTag    sesmSIdTag;
        shastraIdTag    frontSIdTag;
        shastraIdTag    leaderSIdTag;
        shastraIdTag    frontPermTag;
        int outFd;


        ShastraIdTagIn(fd, &sesmSIdTag);
        ShastraIdTagIn(fd, &frontSIdTag);
    ShastraIdTagIn(fd, &leaderSIdTag);
    ShastraIdTagIn(fd, &frontPermTag);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&frontSIdTag, &outFd,
            "collInviteJoinHandler()")){
        case route_DEFAULT:
            collInviteJoinOprn(&sesmSIdTag, &frontSIdTag, &leaderSIdTag,
                &frontPermTag);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollInviteJoinHandler(outFd, &sesmSIdTag, &frontSIdTag,
                &leaderSIdTag, &frontPermTag);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
}
int
collAskJoinHandler(fd)
    int             fd;
{
    shastraIdTag    sesmSIdTag;
    shastraIdTag    frontSIdTag;
    int outFd;

    ShastraIdTagIn(fd, &sesmSIdTag);
    ShastraIdTagIn(fd, &frontSIdTag);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeSesMgrSIdTagToFd(&sesmSIdTag, &outFd,
            "collAskJoinHandler()")){
        case route_DEFAULT:
            collAskJoinOprn(&sesmSIdTag, &frontSIdTag);
        break;
        case route_KERNEL:
        case route_SESMGR:
            putCollAskJoinHandler(outFd, &sesmSIdTag, &frontSIdTag);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOIN);
    showInfo(sbOutMsgBuf);
        return(0);
```

```
}
int
collTellJoinHandler(fd)
    int             fd;
{
    shastraIdTag    sesmSIdTag;
    shastraIdTag    frontSIdTag;
    shastraIdTag    frontPermTag;
    shastraId      *pSId;
    int             outFd;

    ShastraIdTagIn(fd, &sesmSIdTag);
    ShastraIdTagIn(fd, &frontSIdTag);
    ShastraIdTagIn(fd, &frontPermTag);
    cmAckOk(fd);
    cmFlush(fd);

    pSId = krFrSIdTag2SId(frontSIdTag);
    if (pSId == NULL) {
        sprintf(sbOutMsgBuf, "collTellJoinHandler()->Unknown IDTag --
            Aborted\n");
        showInfo(sbOutMsgBuf);
        return(0);
    }
    if (pSId->lIPAddr != kernelShastraId.lIPAddr) {
        if (fMainKernel) {
            outFd = shaKernId2Fd(pSId);
            if (outFd == -1) {
    sprintf(sbOutMsgBuf, "collTellJoinHandler()->Unknown Kernel -- Aborted\
        n");
    showInfo(sbOutMsgBuf);
                return(0);
            }
            putCollTellJoinHandler(outFd, &sesmSIdTag,
                           &frontSIdTag, &frontPermTag);
        } else {
            collTellJoinOprn(&sesmSIdTag, &frontSIdTag,
                        &frontPermTag);
        }
    } else {
        int             outFd;
        outFd = shaFrontId2Fd(pSId);
        if (outFd == -1) {
            sprintf(sbOutMsgBuf, "collTellJoinHandler()->Unknown Front --
                Aborted\n");
            showInfo(sbOutMsgBuf);
            return(0);
        }
        putCollTellJoinHandler(outFd, &sesmSIdTag, &frontSIdTag,
                         &frontPermTag);
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
    showInfo(sbOutMsgBuf);
```

```
        return(0);
}

/*
 * Function
 */
int
putShaKernIdHandler(fd)
    int             fd;
{
    putStringOnChannel(fd, REQ_SET_SHAKERNID, "putShaKernIdHandler()");
    ShastraIdsOut(fd, &shastraKernIds);
    cmFlush(fd);
        return(0);
}


/*
 * Function
 */
int
putShaKernFrIdHandler(fd, pSIdKern)
    int             fd;
    shastraId      *pSIdKern;
{
    shastraIds     *pSIds;
    int             krIndex;

    putStringOnChannel(fd, REQ_SET_SHAKERNFRID, "putShaKernFrIdHandler()");
    cmFlush(fd);
    ShastraIdOut(fd, pSIdKern);
    cmFlush(fd);
    krIndex = locateKernFronts(pSIdKern);
    if (krIndex == -1) {
        fprintf(stderr, "putShaKernFrIdHandler()-> unlocated kernel!\n");
        krIndex = 0;
    }
    pSIds = getKernFrontSIds(pSIdKern);
    ShastraIdsOut(fd, pSIds);
    cmFlush(fd);
    if (debug) {
        outputId(stderr, pSIdKern);
        outputIds(stderr, pSIds);
    }
    cmFlush(fd);
        return(0);
}



/*
 * Function
 */
```

```
int
putShaSesmIdHandler(fd)
    int             fd;
{
    putStringOnChannel(fd, REQ_SET_SHASESMID, "putShaSesmIdHandler()");
    ShastraIdsOut(fd, &shastraSesmIds);
    cmFlush(fd);
        return(0);
}


/*
 * Function
 */
int
putShaSesmFrIdHandler(fd, pSIdTagSesm)
    int             fd;
    shastraIdTag   *pSIdTagSesm;
{
    shastraIdTags  *pSIdTags;
    shastraIdTags  *pPermTags;
    int             smIndex;

    putStringOnChannel(fd, REQ_SET_SHASESMFRID, "putShaSesmFrIdHandler()");
    ShastraIdTagOut(fd, pSIdTagSesm);
    smIndex = locateSesmFronts(pSIdTagSesm);
    if (smIndex == -1) {
        fprintf(stderr, "putShaSesmFrIdHandler()-> unlocated sesMgr!\n");
        smIndex = 0;
    }
    pSIdTags = getSesmFrontSIdTags(pSIdTagSesm);
    ShastraIdTagsOut(fd, pSIdTags);
    pPermTags = getSesmFrontPermTags(pSIdTagSesm);
    ShastraIdTagsOut(fd, pPermTags);
    if (debug) {
        outputIdTag(stderr, pSIdTagSesm);
        outputIdTags(stderr, pSIdTags);
        outputIdTags(stderr, pPermTags);
    }
    cmFlush(fd);
        return(0);
}




/*
 * Function
 */
int
putShaStateHandler(fd)
    int             fd;
{
    int             i;

    putShaKernIdHandler(fd);
    for (i = 0; i < shastraKernIds.shastraIds_len; i++)
        {
        putShaKernFrIdHandler(fd, shastraKernIds.shastraIds_val[i]);
    }
    putShaSesmIdHandler(fd);
    for (i = 0; i < shastraSesmIds.shastraIds_len; i++) {
        putShaSesmFrIdHandler(fd, & shastraSesmIds.shastraIds_val[i]->
            lSIDTag);
    }
        return(0);
}

/*
 * Function
 */
int
putShaStartSysHandler(fd, pSIdCreate)
    int             fd;
    shastraId      *pSIdCreate;
{
    putStringOnChannel(fd, REQ_START_SYSTEM, "putShaStartSysHandler()");
    ShastraIdOut(fd, pSIdCreate);
    if (debug) {
        outputId(stderr, pSIdCreate);
    }
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int
putShaEndSysHandler(fd, pSIdKill)
    int             fd;
    shastraId      *pSIdKill;
{
    putStringOnChannel(fd, REQ_END_SYSTEM, "putShaEndSysHandler()");
    ShastraIdOut(fd, pSIdKill);
    if (debug) {
        outputId(stderr, pSIdKill);
    }
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int
```

```
putShaTerminateHandler(fd)
    int             fd;
{
    putStringOnChannel(fd, REQ_TERMINATE, "putShaTerminateHandler()");
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int
putCollInviteJoinHandler(fd, pSesmIdTag, pFrontIdTag, pLeaderIdTag,
        pFrontPermTag)
    int             fd;
    shastraIdTag   *pSesmIdTag;
    shastraIdTag   *pFrontIdTag;
    shastraIdTag   *pLeaderIdTag;
    shastraIdTag   *pFrontPermTag;
{
    putStringOnChannel(fd, REQ_COLL_INVITEJOIN, "putCollInviteJoinHandler(
        )");
    ShastraIdTagOut(fd, pSesmIdTag);
    ShastraIdTagOut(fd, pFrontIdTag);
    ShastraIdTagOut(fd, pLeaderIdTag);
    ShastraIdTagOut(fd, pFrontPermTag);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int
putCollAskJoinHandler(fd, pSesmIdTag, pFrontIdTag)
    int             fd;
    shastraIdTag   *pSesmIdTag;
    shastraIdTag   *pFrontIdTag;
{
    putStringOnChannel(fd, REQ_COLL_ASKJOIN, "putCollAskJoinHandler()");
    ShastraIdTagOut(fd, pSesmIdTag);
    ShastraIdTagOut(fd, pFrontIdTag);
    cmFlush(fd);
        return(0);
}

/*
 * function() --
 */
int
putCollTellJoinHandler(fd, pSesmIdTag, pFrontIdTag, pFrontPermTag)
    int             fd;
    shastraIdTag   *pSesmIdTag;
    shastraIdTag   *pFrontIdTag;
    shastraIdTag   *pFrontPermTag;
{
    putStringOnChannel(fd, REQ_COLL_TELLJOIN, "putCollTellJoinHandler()");
    ShastraIdTagOut(fd, pSesmIdTag);
    ShastraIdTagOut(fd, pFrontIdTag);
    ShastraIdTagOut(fd, pFrontPermTag);
    cmFlush(fd);
        return(0);
}

/*
 * function() --
 */
int
closedChannelCleanUpHandler(fd)
    int            fd;
{
    switch (shaKernFlags[fd]) {
    case SHAKERNEL:
#ifdef DEBUG
        fprintf(stderr, "closedChannelCleanUpHandler(%d)--kernel
            disconnected!\n", fd);
#endif /* DEBUG */
        quitKernelCleanUpHandler(fd);
        break;
    case SHASESMGR:
#ifdef DEBUG
        fprintf(stderr, "closedChannelCleanUpHandler(%d)--sesmgr
            disconnected!\n", fd);
#endif /* DEBUG */
        quitSesMgrCleanUpHandler(fd);
        break;
    case SHAFRONT:
#ifdef DEBUG
        fprintf(stderr, "closedChannelCleanUpHandler(%d)--front
            disconnected!\n", fd);
#endif /* DEBUG */
        quitFrontCleanUpHandler(fd);
        break;
    default:
#ifdef DEBUG
        fprintf(stderr, "closedChannelCleanUpHandler(%d)--unknown client
            disconnected!\n", fd);
#endif /* DEBUG */
        mplexUnRegisterChannel(fd);
        break;
    }
        return(0);
}

/*
 * Function
 */
int putCollInviteMsgHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COLL_INVITEMSG, "putCollInviteMsgHandler()")
        ;
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int collInviteMsgHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "collInviteMsgHandler()")){
        case route_DEFAULT:
            collInviteMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCollInviteMsgHandler(outFd, &smSIdTag, &toSIdTag,
                &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITEMSG);
```

```
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int putCollInvRespMsgHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
        int fd;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        char *sbMsg;
    {
        putStringOnChannel(fd, REQ_COLL_INVRESPMSG, "putCollInvRespMsgHandler(
            )");
        ShastraIdTagOut(fd, pSmSIdTag);
        ShastraIdTagOut(fd, pToSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        sendDataString(fd, sbMsg);
        cmFlush(fd);
            return(0);
    }

    /*
     * Function
     */
    int collInvRespMsgHandler(fd)
        int fd;
    {
        shastraIdTag    smSIdTag;
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        char *sMsg;
        int outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        sMsg = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collInvRespMsgHandler()")){
            case route_DEFAULT:
                collInvRespMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCollInvRespMsgHandler(outFd, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVRESPMSG);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int putCollInviteStatusHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
        int fd;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        shaULong lStatus;
    {
        putStringOnChannel(fd, REQ_COLL_INVITESTATUS,
            "putCollInviteStatusHandler()");
        ShastraIdTagOut(fd, pSmSIdTag);
        ShastraIdTagOut(fd, pToSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        ShastraULongOut(fd, &lStatus);
        cmFlush(fd);
            return(0);
    }

    /*
     * Function
     */
    int collInviteStatusHandler(fd)
        int fd;
    {
        shastraIdTag    smSIdTag;
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        shaULong        lStatus;
        int outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        ShastraULongIn(fd, &lStatus);
        cmAckOk(fd);
        cmFlush(fd);

        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collInviteStatusHandler()")){
            case route_DEFAULT:
                collInviteStatusReq(pHostMainKern, &smSIdTag, &toSIdTag,
                    &sIdTag, lStatus);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCollInviteStatusHandler(outFd, &smSIdTag, &toSIdTag,
                    &sIdTag, lStatus);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_INVITESTATUS);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int putCollAskJoinMsgHandler(fd, pSmSIdTag, pSIdTag, sbMsg)
        int fd;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pSIdTag;
        char *sbMsg;
    {
        putStringOnChannel(fd, REQ_COLL_ASKJOINMSG, "putCollAskJoinMsgHandler(
            )");
        ShastraIdTagOut(fd, pSmSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        sendDataString(fd, sbMsg);
        cmFlush(fd);
            return(0);
    }

    /*
     * Function
     */
    int collAskJoinMsgHandler(fd)
        int fd;
    {
        shastraIdTag    smSIdTag;
        shastraIdTag    sIdTag;
        char *sMsg;
        int outFd;

        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        sMsg = cmReceiveString(fd);
        cmAckOk(fd);
        cmFlush(fd);

        switch(routeSesMgrSIdTagToFd(&smSIdTag, &outFd,
                "collAskJoinMsgHandler()")){
```

```
        case route_DEFAULT:
            collAskJoinMsgReq(pHostMainKern, &smSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_SESMGR:
            putCollAskJoinMsgHandler(outFd, &smSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOINMSG);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int putCollAskJnRespMsgHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pSmSIdTag;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COLL_ASKJNRESPMSG,
        "putCollAskJnRespMsgHandler()");
    ShastraIdTagOut(fd, pSmSIdTag);
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int collAskJnRespMsgHandler(fd)
    int fd;
{
    shastraIdTag    smSIdTag;
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &smSIdTag);
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
```

```
        cmFlush(fd);

        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collAskJnRespMsgHandler()")){
            case route_DEFAULT:
                collAskJnRespMsgReq(pHostMainKern, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCollAskJnRespMsgHandler(outFd, &smSIdTag, &toSIdTag,
                    &sIdTag, sMsg);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNRESPMSG);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int putCollAskJnStatusHandler(fd, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
        int fd;
        shastraIdTag *pSmSIdTag;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        shaULong lStatus;
    {
        putStringOnChannel(fd, REQ_COLL_ASKJNSTATUS, "putCollAskJnStatusHandler
            ()");
        ShastraIdTagOut(fd, pSmSIdTag);
        ShastraIdTagOut(fd, pToSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        ShastraULongOut(fd, &lStatus);
        cmFlush(fd);
            return(0);
    }

    /*
     * Function
     */
    int collAskJnStatusHandler(fd)
        int fd;
    {
        shastraIdTag    smSIdTag;
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
        shaULong        lStatus;
        int outFd;
        ShastraIdTagIn(fd, &smSIdTag);
        ShastraIdTagIn(fd, &toSIdTag);
        ShastraIdTagIn(fd, &sIdTag);
        ShastraULongIn(fd, &lStatus);
        cmAckOk(fd);
        cmFlush(fd);

        switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
                "collAskJnStatusHandler()")){
            case route_DEFAULT:
                collAskJnStatusReq(pHostMainKern, &smSIdTag, &toSIdTag,
                    &sIdTag, lStatus);
            break;
            case route_KERNEL:
            case route_FRONT:
                putCollAskJnStatusHandler(outFd, &smSIdTag, &toSIdTag,
                    &sIdTag, lStatus);
            break;
            case route_ERROR:
            default:
            break;
        }
        sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNSTATUS);
        showInfo(sbOutMsgBuf);
            return(0);
    }

    /*
     * Function
     */
    int putCommMsgTextHandler(fd, pToSIdTag, pSIdTag, sbMsg)
        int fd;
        shastraIdTag *pToSIdTag;
        shastraIdTag *pSIdTag;
        char *sbMsg;
    {
        putStringOnChannel(fd, REQ_COMM_MSGTEXT, "putCommMsgTextHandler()");
        ShastraIdTagOut(fd, pToSIdTag);
        ShastraIdTagOut(fd, pSIdTag);
        sendDataString(fd, sbMsg);
        cmFlush(fd);
            return(0);
    }

    /*
     * Function
     */
    int commMsgTextHandler(fd)
        int fd;
    {
        shastraIdTag    toSIdTag;
        shastraIdTag    sIdTag;
```

```
	char *sMsg;
	int outFd;

	ShastraIdTagIn(fd, &toSIdTag);
	ShastraIdTagIn(fd, &sIdTag);
	sMsg = cmReceiveString(fd);
	cmAckOk(fd);
	cmFlush(fd);

	switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
			"commMsgTextHandler()")){
		case route_DEFAULT:
			commMsgTextReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
		break;
		case route_KERNEL:
		case route_FRONT:
			putCommMsgTextHandler(outFd, &toSIdTag, &sIdTag, sMsg);
		break;
		case route_ERROR:
		default:
		break;
	}
	sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXT);
	showInfo(sbOutMsgBuf);
		return(0);
}

/*
 * Function
 */
int putCommMsgTextFileHandler(fd, pToSIdTag, pSIdTag, sbMsg)
	int fd;
	shastraIdTag *pToSIdTag;
	shastraIdTag *pSIdTag;
	char *sbMsg;
{
	putStringOnChannel(fd, REQ_COMM_MSGTEXTFILE, "putCommMsgTextFileHandler
		()");
	ShastraIdTagOut(fd, pToSIdTag);
	ShastraIdTagOut(fd, pSIdTag);
	sendDataString(fd, sbMsg);
	cmFlush(fd);
		return(0);
}

/*
 * Function
 */
int commMsgTextFileHandler(fd)
	int fd;
{
	shastraIdTag	toSIdTag;
	shastraIdTag	sIdTag;
	char *sMsg;
	int outFd;

	ShastraIdTagIn(fd, &toSIdTag);
	ShastraIdTagIn(fd, &sIdTag);
	sMsg = cmReceiveString(fd);
	cmAckOk(fd);
	cmFlush(fd);

	switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
			"commMsgTextFileHandler()")){
		case route_DEFAULT:
			commMsgTextFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
		break;
		case route_KERNEL:
		case route_FRONT:
			putCommMsgTextFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
		break;
		case route_ERROR:
		default:
		break;
	}
	sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXTFILE);
	showInfo(sbOutMsgBuf);
		return(0);
}

/*
 * Function
 */
int putCommMsgAudioHandler(fd, pToSIdTag, pSIdTag, sbMsg)
	int fd;
	shastraIdTag *pToSIdTag;
	shastraIdTag *pSIdTag;
	char *sbMsg;
{
	putStringOnChannel(fd, REQ_COMM_MSGAUDIO, "putCommMsgAudioHandler()");
	ShastraIdTagOut(fd, pToSIdTag);
	ShastraIdTagOut(fd, pSIdTag);
	sendDataString(fd, sbMsg);
	cmFlush(fd);
		return(0);
}

/*
 * Function
 */
int commMsgAudioHandler(fd)
	int fd;
{
	shastraIdTag	toSIdTag;
	shastraIdTag	sIdTag;
	char *sMsg;
```

```
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "commMsgAudioHandler()")){
        case route_DEFAULT:
            commMsgAudioReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgAudioHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIO);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int putCommMsgAudioFileHandler(fd, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGAUDIOFILE,
        "putCommMsgAudioFileHandler()");
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int commMsgAudioFileHandler(fd)
    int fd;
{
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;

    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "commMsgAudioFileHandler()")){
        case route_DEFAULT:
            commMsgAudioFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgAudioFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIOFILE);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int putCommMsgVideoHandler(fd, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGVIDEO, "putCommMsgVideoHandler()");
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int commMsgVideoHandler(fd)
    int fd;
{
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "commMsgVideoHandler()")){
        case route_DEFAULT:
            commMsgVideoReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgVideoHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEO);
    showInfo(sbOutMsgBuf);
        return(0);
}

/*
 * Function
 */
int putCommMsgVideoFileHandler(fd, pToSIdTag, pSIdTag, sbMsg)
    int fd;
    shastraIdTag *pToSIdTag;
    shastraIdTag *pSIdTag;
    char *sbMsg;
{
    putStringOnChannel(fd, REQ_COMM_MSGVIDEOFILE,
        "putCommMsgVideoFileHandler()");
    ShastraIdTagOut(fd, pToSIdTag);
    ShastraIdTagOut(fd, pSIdTag);
    sendDataString(fd, sbMsg);
    cmFlush(fd);
        return(0);
}

/*
 * Function
 */
int commMsgVideoFileHandler(fd)
    int fd;
{
    shastraIdTag    toSIdTag;
    shastraIdTag    sIdTag;
    char *sMsg;
    int outFd;
    ShastraIdTagIn(fd, &toSIdTag);
    ShastraIdTagIn(fd, &sIdTag);
    sMsg = cmReceiveString(fd);
    cmAckOk(fd);
    cmFlush(fd);

    switch(routeFrontSIdTagToFd(&toSIdTag, &outFd,
            "commMsgVideoFileHandler()")){
        case route_DEFAULT:
            commMsgVideoFileReq(pHostMainKern, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_KERNEL:
        case route_FRONT:
            putCommMsgVideoFileHandler(outFd, &toSIdTag, &sIdTag, sMsg);
        break;
        case route_ERROR:
        default:
        break;
    }
    sprintf(sbOutMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEOFILE);
    showInfo(sbOutMsgBuf);
        return(0);
}
```

```
/*************************************************************************
    ***/
/*************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*************************************************************************
    ***/
/*************************************************************************
    ***/
/*
 * kernelfind.c - find the master kernel
 *
 */

#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>
#include <X11/Intrinsic.h>

#define RESPORT  9999
#define MAINPORT 9998
#define NAMELEN 128
char *myhostname = NULL;
static int kernnameserver(char *, int *, unsigned long *);

char *MasterKernelName(char *myhostname)
{
    int ssock;
    struct timeval timeout;
    int i,result;
    fd_set iReadMask,iWriteMask, iExcepnMask;
    char buf[NAMELEN];
    int res;
    char *told;

    if ((told = getenv("MASTERKERNEL")) != NULL)
    {
        return(told);
    }

    memset(buf,0,NAMELEN);
    ssock = ntBroadcastServer(RESPORT);
    res = ntBroadcast(MAINPORT,myhostname,strlen(myhostname));

    FD_ZERO( &iReadMask);
    FD_ZERO( &iWriteMask);
    FD_ZERO( &iExcepnMask);
    FD_SET(ssock, &iReadMask);
    timeout.tv_sec = 3;
    timeout.tv_usec = 0;

    if ((result = select(ssock+1, (fd_set *)&iReadMask,
               (fd_set *)&iWriteMask, (fd_set *)&iExcepnMask,&timeout))
                   <= 0)
    {
     return (NULL);
    }

    if (FD_ISSET(ssock, &iReadMask))
    {
        read(ssock, buf, NAMELEN);
        close(ssock);
        return(strdup(buf));
    }
    return(NULL);
}

int SetupKernelNameServer(XtAppContext xac, char *myname)
{
    int ssock;

    ssock = ntBroadcastServer(MAINPORT);
    myhostname = strdup(myname);
    XtAppAddInput(xac, ssock, (XtPointer)XtInputReadMask,
        (XtInputCallbackProc)kernnameserver ,NULL);
    XtAppAddInput(xac, ssock, (XtPointer)XtInputExceptMask,
        (XtInputCallbackProc)kernnameserver ,NULL);
}

int kernnameserver(char *arg, int *pfd, unsigned long *plId)
{
    char buf[NAMELEN];
    int l;
    int res;
    int fd;

    fd = *pfd;
    memset(buf, 0, NAMELEN);
    l = read(fd, buf, NAMELEN);
    res = ntBroadcast(RESPORT,myhostname,strlen(myhostname));
    return(res);
}

int ntBroadcastServer(int port)
{
     int isocket;
     struct sockaddr_in sa;
     int iOption;
     int res;

     if ((isocket = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
     {
         perror("socket()");
         return(-1);
     }

     sa.sin_family = AF_INET;
     sa.sin_addr.s_addr = INADDR_ANY;
     sa.sin_port = htons(port);

     if (bind(isocket, (struct sockaddr *)&sa, sizeof(sa)) != 0)
     {
         perror("bind():");
         close(isocket);
         return(-1);
     }

     iOption = 1;
     if (setsockopt(isocket, SOL_SOCKET, SO_REUSEADDR,
                 (const char *)&iOption, sizeof(iOption)) == -1)
     {
       perror("setsockopt() SOL_SOCKET, SO_REUSEADDR");
       close(isocket);
       return(-1);;
     }
     iOption = 1;
     if (setsockopt(isocket, SOL_SOCKET, SO_BROADCAST,
                 (const char *)&iOption, sizeof(iOption)) == -1)
     {
       perror("setsockopt() SOL_SOCKET, SO_BROADCAST");
       close(isocket);
       return(-1);;
     }

     return(isocket);
}
```

```
int ntBroadcast(int port, char *buf, int numbytes)
{
    int res;
    int sock;
    struct sockaddr_in sa;
    struct hostent *mhost;
    char hostname[255];
    int value;
    int status;

    sock = socket(AF_INET, SOCK_DGRAM, 0);

    value = 1;
    status = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (const char *)&
        value, sizeof(int));
    if (status == -1)
    {
      perror("setsockopt");
      exit(1);
    }

    gethostname(hostname, 255);
    if ((mhost = gethostbyname(hostname)) == NULL)
    {
        fprintf(stderr, "unknown host %s\n", "localhost");
        close(sock);
        return(-1);
    }
    memcpy((char *)&sa.sin_addr, mhost->h_addr, mhost->h_length);
    sa.sin_family = AF_INET;
    /*sa.sin_addr.s_addr = sa.sin_addr.s_addr | 0x000000ff ;*/
    /* well we have a broadcast net here at Purdue But NTT has a
       multicast net.  Its weird! */
    /*sa.sin_addr.s_addr = 0xe0000001;*/
    /* for a 8 bit subnet */
    sa.sin_addr.s_addr = sa.sin_addr.s_addr | 0x000000ff ;
    fprintf(stderr, "Addr %x\n", sa.sin_addr.s_addr);
    sa.sin_port = htons(port);
    res = sendto(sock, buf, numbytes, 0, (struct sockaddr *)&sa, sizeof(sa)
        );
    if (res < 0)
    {
      perror("ntBroadcast");
    }
    close(sock);
    return(0);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <fcntl.h>
#include <nlist.h>
#include <unistd.h>
#ifdef SHASTRA4SUN5
#include <stdlib.h>
#endif

/*
 * code to get load avergae.. sadly, /dev/kmem is not readable anymore
 */

static void         getLoadError();

#ifdef WANTTHIS

#ifdef SHASTRA4IRIS

#define KERNEL_FILE "/unix"
#define KERNEL_MEMFILE "/dev/kmem"
#define LOADAVGNDX 0
#define KERNEL_LOAD_VARIABLE "avenrun"
extern void     exit();
static struct nlist loadAvgNmList[] = {
    {KERNEL_LOAD_VARIABLE},
    {NULL}
};
static          kernelMemFD;
static long     loadAvgSeekOffset;
```

```
void
getLoadAvg(pLoadAvg)
    double          *pLoadAvg;
{
    long            temp;

    if (loadAvgSeekOffset == 0) {
        nlist(KERNEL_FILE, loadAvgNmList);
        if (loadAvgNmList[LOADAVGNDX].n_type == 0 ||
            loadAvgNmList[LOADAVGNDX].n_value == 0) {
            getLoadError("cannot get name list from", KERNEL_FILE);
            *pLoadAvg = 0.0;
            return;
        }
        loadAvgSeekOffset = loadAvgNmList[LOADAVGNDX].n_value;
    }
    kernelMemFD = open(KERNEL_MEMFILE, O_RDONLY);
    if (kernelMemFD < 0) {
        getLoadError("cannot open", KERNEL_MEMFILE);
        *pLoadAvg = 0.0;
        return;
    }
    lseek(kernelMemFD, loadAvgSeekOffset, 0);
    (void) read(kernelMemFD, (char *) &temp, sizeof(long));
    close(kernelMemFD);
    *pLoadAvg = (double) temp / 1024.0;
    return;
}

#endif              /* SHASTRA4IRIS */

#ifdef SHASTRA4SUN4

#define KERNEL_FILE "/vmunix"
#define KERNEL_MEMFILE "/dev/kmem"
#define LOADAVGNDX 0
#define KERNEL_LOAD_VARIABLE "_avenrun"
extern void     exit();
static struct nlist loadAvgNmList[] = {
    {KERNEL_LOAD_VARIABLE},
    {NULL}
};
static          kernelMemFD;
static long     loadAvgSeekOffset;


void
getLoadAvg(pLoadAvg)
    double          *pLoadAvg;
{
    long            temp;
    if (loadAvgSeekOffset == 0) {
        nlist(KERNEL_FILE, loadAvgNmList);
        if (loadAvgNmList[LOADAVGNDX].n_type == 0 ||
            loadAvgNmList[LOADAVGNDX].n_value == 0) {
            getLoadError("cannot get name list from", KERNEL_FILE);
            *pLoadAvg = 0.0;
            return;
        }
        loadAvgSeekOffset = loadAvgNmList[LOADAVGNDX].n_value;
    }
    kernelMemFD = open(KERNEL_MEMFILE, O_RDONLY);
    if (kernelMemFD < 0) {
        getLoadError("cannot open", KERNEL_MEMFILE);
        *pLoadAvg = 0.0;
        return;
    }
    lseek(kernelMemFD, loadAvgSeekOffset, 0);
    (void) read(kernelMemFD, (char *) &temp, sizeof(long));
    close(kernelMemFD);
    *pLoadAvg = (double) temp / (1 << 8);
    return;
}

#endif              /* SHASTRA4SUN4 */


#endif              /* WANTTHIS */

void
getLoadAvg(pLoadAvg)
    double        *pLoadAvg;
{
    char           tmpFilBuf[32];
    char           tmpCmdBuf[64];
    FILE          *loadFile;

    sprintf(tmpFilBuf, "/tmp/#load%d", (int)getpid());
    sprintf(tmpCmdBuf, "uptime | /usr/bin/awk '{print $10}' > %s",
        tmpFilBuf);
    if (system(tmpCmdBuf) != 0) {
        perror("getLoadAvg()-- system()");
        *pLoadAvg = 0.0;
        return;
    }
    if (access(tmpFilBuf, R_OK) == -1) {
        perror("getLoadAvg() -- access()");
        *pLoadAvg = 0.0;
        return;
    }
    if ((loadFile = fopen(tmpFilBuf, "r")) == NULL) {
        perror("getLoadAvg() -- fopen()");
        *pLoadAvg = 0.0;
        return;
        }
        fscanf(loadFile, "%lf", pLoadAvg);
        fclose(loadFile);
        unlink(tmpFilBuf);
        return;

    }

    static void
    getLoadError(str1, str2)
        char            *str1, *str2;
    {
        fprintf(stderr, "getLoad(): %s %s\n", str1, str2);
        perror("getLoad()");
    }
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/time.h>

#include <shastra/utils/list.h>
#include <shastra/network/asyncIO.h>

#define STANDALONEnn

struct list    *aIOInList;
struct list    *aIOOutList;
struct list    *aIOReplayOutList;

static void     (*aIOReadHandler) (Prot2(aIOControl*,char*));
static char     *aIOReadArg;
static void     (*aIOWriteHandler) (Prot2(aIOControl*,char*));
static char     *aIOWriteArg;
static void     handleAIO(Prot1(aio_result_t *));

void
clearPendingAIO()
{
    struct list_node *node;
    aIOControl     *pAIOCntl;

    while (aIOInList->head != NULL) {
        node = aIOInList->head;
        pAIOCntl = (aIOControl *) node->data;
        if (aiocancel(&pAIOCntl->resultAIO) == -1) {
            fprintf(stderr, "clearPendingAIO()->couldn't cancel %lx\n",
                &pAIOCntl->resultAIO);
        }
        listDeleteThis(aIOInList, node);
        free(pAIOCntl->buf);
        free(pAIOCntl);
        free(node);
    }
    while (aIOOutList->head != NULL) {
        node = aIOOutList->head;
        pAIOCntl = (aIOControl *) node->data;
        if (aiocancel(&pAIOCntl->resultAIO) == -1) {
            fprintf(stderr, "clearPendingAIO()->couldn't cancel %lx\n",
                &pAIOCntl->resultAIO);
        }
        listDeleteThis(aIOOutList, node);
        free(pAIOCntl->buf);
        free(pAIOCntl);
        free(node);
    }
    while (aIOReplayOutList->head != NULL) {/*no async in this*/
        node = aIOReplayOutList->head;
        pAIOCntl = (aIOControl *) node->data;
        listDeleteThis(aIOReplayOutList, node);
        free(pAIOCntl->buf);
        free(pAIOCntl);
        free(node);
    }
}

void
registerAIOReadHandler(func, arg)
    void            (*func) ();
    char            *arg;
{
    aIOReadHandler = func;
    aIOReadArg = arg;
}

void
registerAIOWriteHandler(func, arg)
    void            (*func) ();
    char            *arg;
{
    aIOWriteHandler = func;
    aIOWriteArg = arg;
}

void
sigIOHandler()
{
    aio_result_t   *resultAIO;
    aio_result_t   *aiowait();
    struct timeval  timeout;
    static int      fFirst = 1;
    static int      ctr;

#ifdef DEBUG
    fprintf(stderr, "In sigIOHandler call %d\n", ctr);
#endif              /* DEBUG */

    memset(&timeout, 0, sizeof(struct timeval));
    while ((resultAIO = aiowait(&timeout)) != 0) {
        if (resultAIO == (aio_result_t *) - 1) {
            if (fFirst) {
                perror("aiowait()");
            }
            break;
        } else {
#ifdef DEBUG
            fprintf(stderr, "resultAIO = %lx\n", resultAIO);
#endif              /* DEBUG */
            handleAIO(resultAIO);
        }
        fFirst = 0;
    }
    /* poll returned null */
#ifdef DEBUG
    fprintf(stderr, "Out sigIOHandler call %d\n", ctr++);
#endif              /* DEBUG */
}

static void
handleAIO(resultAIO)
    aio_result_t   *resultAIO;
{
    aio_result_t   *aiowait();
    aIOControl     *pAIOInCntl, *pAIOOutCntl;

#ifdef DEBUG
    fprintf(stderr, "In handleAIO\n");
#endif              /* DEBUG */
    if (aIOInList->head != NULL) {
        pAIOInCntl = (aIOControl *) aIOInList->head->data;
    } else {
        pAIOInCntl = NULL;
    }
    if (aIOOutList->head != NULL) {
        pAIOOutCntl = (aIOControl *) aIOOutList->head->data;
    } else {
        pAIOOutCntl = NULL;
    }
    if (pAIOInCntl && (resultAIO == &pAIOInCntl->resultAIO)) {
        if (resultAIO->aio_return == -1) {
            extern int      errno;
            errno = resultAIO->aio_errno;
            perror("aiowait()->read()");
        } else {
#ifdef DEBUG
            fprintf(stderr, "handleAIO()-> Read()-> %d of %d of %lx\n",
                resultAIO->aio_return, pAIOInCntl->bufSize, resultAIO);
#endif /*DEBUG*/
            pAIOInCntl->bufSize = resultAIO->aio_return;
            if (aIOReadHandler != NULL) {
                (*aIOReadHandler) (pAIOInCntl,aIOReadArg);
            }
        }
    } else if (pAIOOutCntl && (resultAIO == &pAIOOutCntl->resultAIO)) {
        if (resultAIO->aio_return == -1) {
            extern int      errno;
            errno = resultAIO->aio_errno;
            perror("aiowait()->write()");
        } else {
#ifdef DEBUG
            fprintf(stderr, "handleAIO()-> Write()-> %d of %d of %lx\n",
                resultAIO->aio_return, pAIOOutCntl->bufSize, resultAIO);
#endif /*DEBUG*/
            if (aIOWriteHandler != NULL) {
                (*aIOWriteHandler) (pAIOOutCntl,aIOWriteArg);
            }
        }
    } else {
        fprintf(stderr, "handleAIO()-> non-requested return\t");
        if (pAIOInCntl) {
            fprintf(stderr, "In head is %lx\t", &pAIOInCntl->resultAIO);
        }
        if (pAIOOutCntl) {
            fprintf(stderr, "Out head is %lx\t", &pAIOOutCntl->resultAIO);
        }
        fprintf(stderr, "\n");
    }
#ifdef DEBUG
    fprintf(stderr, "Out handleAIO\n");
#endif              /* DEBUG */
}

void
setupSigIOHandler(func)
    void            (*func) ();
{
#if defined SHASTRA4IRIS  || defined SHASTRA4SUN5
    sigset(SIGIO, func);
#else
#ifdef SHASTRA4HP
    signal(SIGIO, func);
#else/* SHASTRA4SUN4 */
    struct sigvec   vec;
    /* Set up SIGIO handler to flush output */
    vec.sv_handler = func;
    vec.sv_mask = 0;
    vec.sv_flags = 0;
    (void) sigvec(SIGIO, &vec, (struct sigvec *) NULL);
#endif              /* SHASTRA4IRIS */
#endif
}


#if defined SHASTRA4IRIS || defined SHASTRA4HP

aio_result_t   *
aiowait()
{
}

int
aioread()
{
}

int
aiowrite()
{
}

int
aiocancel()
{
}

#endif              /* SHASTRA4IRIS */


#ifdef STANDALONE
#define BUFSIZE 2000000

int             inFd = 0;
int             outFd = 1;

main()
{
    int             tmp;
    void            testAIOReadHandler();
    void            testAIOWriteHandler();

    if ((outFd = open("/tmp/try", O_WRONLY | O_TRUNC | O_CREAT)) < 0) {
        perror("open()->/tmp/try");
        exit(-1);
    }
    if ((inFd = open("/tmp/try2", O_RDONLY)) < 0) {
            perror("open()->/tmp/try2");
            exit(-1);
        }
        setupSigIOHandler(sigIOHandler);
        registerAIOReadHandler(testAIOReadHandler, NULL);
        registerAIOWriteHandler(testAIOWriteHandler, NULL);

        aIOInList = listMakeNew();
        aIOOutList = listMakeNew();

        testAIOReadHandler(NULL, NULL);
        fprintf(stderr, "Waiting for aio to end\n");
        scanf("%d", &tmp);
    }


    void
    testAIOReadHandler(pAIOCntl,arg)
        aIOControl     *pAIOCntl;
        char *arg;
    {
        static int      fNotFirst = 0;
        struct list_node *node;
        aIOControl     *pAIOCntlNew;
        aIOControl     *pAIOCntlOld;

    #ifdef DEBUG
        fprintf(stderr, "testAIOReadHandler, fNot = %d\n", fNotFirst);
    #endif              /* DEBUG */
        if (fNotFirst) {
            /* advance read ptr in input */
            lseek(inFd, pAIOCntl->bufSize, SEEK_CUR);
            pAIOCntlOld = (aIOControl *) aIOInList->head->data;
            if (pAIOCntl != pAIOCntlOld) {
                fprintf(stderr, "testAIOReadHandler()->bad pAIOCntl %lx, %lx\n"
                    ,
                    pAIOCntl, pAIOCntlOld);
            }
            /* this read is done, remove */
            node = aIOInList->head;
            listDeleteThis(aIOInList, node);
            free(node);
            node = NULL;
            if (pAIOCntl->resultAIO.aio_return == 0) {
                /* last read returns 0 , all read jobs done */
                return;
            }
            if (aIOOutList->head == NULL) {
                /* out queue is empty, initiate a write */
                node = listMakeNewNode();
                pAIOCntlNew = (aIOControl *) malloc(sizeof(aIOControl));
                memset(pAIOCntlNew, 0, sizeof(aIOControl));
                pAIOCntlNew->buf = pAIOCntl->buf;
            pAIOCntlNew->bufSize = pAIOCntl->bufSize;
            node->data = (char *) pAIOCntlNew;
            free(pAIOCntl);
            pAIOCntl = NULL;

            listInsertAtTail(aIOOutList, node);
#ifdef DEBUG
            fprintf(stderr, "Init'g Write\t");
#endif              /* DEBUG */
            if (aiowrite(outFd, pAIOCntlNew->buf, pAIOCntlNew->bufSize,
                0, SEEK_CUR, &pAIOCntlNew->resultAIO) < 0) {
                perror("aiowrite()");
            }
#ifdef DEBUG
            fprintf(stderr, "Init'd resultAIO = %lx\n",
                &pAIOCntlNew->resultAIO);
#endif              /* DEBUG */
        } else {
            /* write in progress.. add to queue */
            node = listMakeNewNode();
            pAIOCntlNew = (aIOControl *) malloc(sizeof(aIOControl));
            memset(pAIOCntlNew, 0, sizeof(aIOControl));
            pAIOCntlNew->buf = pAIOCntl->buf;
            pAIOCntlNew->bufSize = pAIOCntl->bufSize;
            node->data = (char *) pAIOCntlNew;

            free(pAIOCntl);
            pAIOCntl = NULL;

            listInsertAtTail(aIOOutList, node);
        }
    }
    node = listMakeNewNode();
    pAIOCntl = (aIOControl *) malloc(sizeof(aIOControl));
    memset(pAIOCntl, 0, sizeof(aIOControl));
    pAIOCntl->buf = (char *) malloc(BUFSIZE);
    pAIOCntl->bufSize = BUFSIZE;

    node->data = (char *) pAIOCntl;
    listInsertAtTail(aIOInList, node);

    fNotFirst = 1;
#ifdef DEBUG
    fprintf(stderr, "Init'g Read\t");
#endif              /* DEBUG */
    if (aioread(inFd, pAIOCntl->buf, pAIOCntl->bufSize, 0, SEEK_CUR,
            &pAIOCntl->resultAIO) < 0) {
        perror("aioread()");
    }
#ifdef DEBUG
    fprintf(stderr, "Init'd resultAIO = %lx\n",
        &pAIOCntl->resultAIO);
    fprintf(stderr, "Out testAIOReadHandler\n");
```

```
#endif              /* DEBUG */
}

void
testAIOWriteHandler(pAIOCntl,arg)
    aIOControl     *pAIOCntl;
    char *arg;
{
    struct list_node *node;
    aIOControl     *pAIOCntlOld;

#ifdef DEBUG
    fprintf(stderr, "In testAIOWriteHandler\n");
#endif              /* DEBUG */
    pAIOCntlOld = (aIOControl *) aIOOutList->head->data;
    if (pAIOCntl != pAIOCntlOld) {
        fprintf(stderr, "testAIOWriteHandler()->bad pAIOCntl %lx, %lx\n",
            pAIOCntl, pAIOCntlOld);
    }
    node = aIOOutList->head;
    /* advance write ptr in output */
    lseek(outFd, pAIOCntl->resultAIO.aio_return, SEEK_CUR);
    /* this write is done, remove from list */
    listDeleteThis(aIOOutList, node);

    free(pAIOCntl->buf);
    free(pAIOCntl);
    free(node);
    node = NULL;
    pAIOCntl = NULL;

    if (aIOOutList->head != NULL) {
        node = aIOOutList->head;
        pAIOCntl = (aIOControl *) node->data;
#ifdef DEBUG
        fprintf(stderr, "Init'g Write\t");
#endif              /* DEBUG */
        if (aiowrite(outFd, pAIOCntl->buf, pAIOCntl->bufSize, 0, SEEK_CUR,
                &pAIOCntl->resultAIO) < 0) {
            perror("aiowrite()");
        }
#ifdef DEBUG
        fprintf(stderr, "Init'd resultAIO = %lx\n",
            &pAIOCntl->resultAIO);
#endif              /* DEBUG */
    }
#ifdef DEBUG
    fprintf(stderr, "Out testAIOWriteHandler\n");
#endif              /* DEBUG */
}

#endif              /* STANDALONE */
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <errno.h>

#include <shastra/utils/list.h>
#include <shastra/utils/hash.h>

#include <shastra/datacomm/shastraIdH.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/mplexP.h>

extern char    *readString(Prot1(int));

#define DEBUGxx

/*
 * hostSendRawRequest()
 */
int
hostSendRawRequest(pHost, req)
     hostData       *pHost;
     char           *req;
{
    int retVal;

    if((pHost == NULL) || (pHost->fStatus == shaError)){
      return -1;
    }
    retVal = cmSendString(pHost->fdSocket, req);
    if(retVal == -1){
      pHost->fStatus = shaError;
    }
    return retVal;
}

/*
 * hostSendQueuedRequest()
 */
int
hostSendQueuedRequest(pHost, req, arg)
     hostData       *pHost;
     char           *req;
     char           *arg;
{
    int retVal;

    if((pHost == NULL) || (pHost->fStatus == shaError)){
      return -1;
    }
    hostQueueHostRequest(pHost, req, arg);
    retVal = cmSendString(pHost->fdSocket, req);
    if(retVal == -1){
      pHost->fStatus = shaError;
    }
    return retVal;
}

/*
 * hostSendMatchedRequest() -- NOT COMPLETE
 */
int
hostSendMatchedRequest(pHost, req, arg)
     hostData       *pHost;
     char           *req;
     char           *arg;
{
    int retVal;

    if((pHost == NULL) || (pHost->fStatus == shaError)){
      return -1;
    }
    hostQueueHostRequest(pHost, req, arg);
    retVal = cmSendString(pHost->fdSocket, req);
    if(retVal == -1){
      pHost->fStatus = shaError;
    }
    return retVal;
}

/*
```

```
 * hostQueueHostRequest()
 */
void
hostQueueHostRequest(pHost, req, arg)
     hostData       *pHost;
     char           *req;
     char           *arg;   /* use this to store info needed on return */
{
  struct list_node *tmp_node;
  hostRequest    *hReq;

  hReq = (hostRequest *) malloc(sizeof(hostRequest));
  tmp_node = listMakeNewNode();
  hReq->request = req;
  hReq->arg = arg;
  tmp_node->data = (char *) hReq;
  listInsertAtTail(pHost->sendList, tmp_node);
#ifdef DEBUG
  fprintf(stderr, "hostQueueHostRequest()->inserted %s on %ld!\n",
      req, pHost);
#endif              /* DEBUG */
}
/*
 * hostMapFD2Host(pHostList,fd)
 */
hostData       *
hostMapFD2Host(pHostList, fd)
     struct list    *pHostList;
     int             fd;
{
  struct list_node *tmp_node;
  hostData       *pHost;

  for (tmp_node = pHostList->head; tmp_node != NULL; tmp_node = tmp_node->
      next) {
    pHost = (hostData *) tmp_node->data;
    if (pHost->fdSocket == fd) {
      return (pHost);
    }
  }
  return (NULL);
}

int
shaClientHandler(fd, arg, dummy)
     int             fd;
     char           *arg;
     unsigned long *dummy;
{
  int             fFound, i;
  char           *buf;
  hostRequest    *hReq;
  char           *req;
  hostData       *pHost;
  shaCmdData     *pCmdData;
  cmCommand      *pCmds;    /* the outbound cmd table */
  cmCommand      *pCmdsIn;/* the inbound cmd table */
  struct cmCommand *pCmd;
  struct he      *phe;
  struct list_node *node;

  /* inbounds can occur in 2 places.. when req pending/ not req pending */
  pCmdData = mplexTab[fd].pCmdData;
  pCmds = pCmdData->pCmdTab;
  pCmdsIn = pCmdData->pCmdTabIn;
  pHost = mplexTab[fd].pHost;
/*
  pHost = hostMapFD2Host(pCmdData->hostList, fd);
  */
  if (pHost == NULL) {
    fprintf(stderr, "shaClientHandler()->No Host Data for Connection!\n");
    return -1;
  }
  buf = cmReceiveString(fd);
  if (buf == NULL) {
    fprintf(stderr, "shaClientHandler(%d)->Peer %ld (%s) closed connection\
        n",
        fd, pHost->lSIDTag, (pHost->pSId?pHost->pSId->nmHost:"host"));
    if(mplexErrHandler){
      (*mplexErrHandler) (fd);
    }
    else{
      mplexUnRegisterChannel(fd);
    }
    pHost->fStatus = shaError;
    return -1;
  } else {
    int              n = strlen(buf);
    int              fBlank = 1;

    for (i = 0; i < n; i++) {
      if (!isspace(buf[i])) {
    fBlank = 0;
    break;
      }
    }
    if (fBlank) {   /* blank string.. avoid!! */
      free(buf);
      return;
    }
#ifdef DEBUG
    fprintf(stderr, "shaClientHandler()->Read %d (%s)\n",
        strlen(buf), buf);
#endif              /* DEBUG */
    if (pHost->sendList->head == NULL) {
      /* maybe this is an inbound command! */
      int             retVal;
      retVal = cmNewSearchNExecute(fd, buf, pCmdData->htCmdsIn, arg);
      /*
       * retVal =  cmSearchNExecute(fd,buf, pCmdsIn,
       * pCmdData->nCmdsIn,arg);
       */
      if (retVal == -1) {
    fprintf(stderr, "shaClientHandler()->Unintelligible / Unsolicited Input
        : %s!\n"
        ,buf);
      }
      free(buf);
      return retVal;
    }
    /* read ACK or ERROR .. */
    if (strcmp(buf, ERROR_STRING) == 0) {
      /* ERROR -- message */
      hReq = (hostRequest *) pHost->sendList->head->data;
      req = hReq->request;
      fprintf(stderr, "shaClientHandler()->Error On %s!\n", req);
      node = pHost->sendList->head;
      listDeleteThis(pHost->sendList, node);
      free(buf);
      free(hReq);
      free(node);
      return -1;
    } else if (strcmp(buf, ACK_STRING) == 0) {
      /*
       * ACK -- look in queue for that fd(??) and know what
       * response is for
       */
      hReq = (hostRequest *) pHost->sendList->head->data;
      req = hReq->request;
#ifdef WANT
      fFound = 0;
      for (i = 0; i < pCmdData->nCmds; i++) {
    if (strcmp(pCmds[i].command, req) == 0) {
      fFound = 1;
#ifdef DEBUG
      fprintf(stderr, "%s\n", pCmds[i].helpmsg);
#endif              /* DEBUG */
      (*pCmds[i].function) (fd, (char *) hReq->arg);
      break;
    }
      }
      if (!fFound) {
    fprintf(stderr, "shaClientHandler()->Unknown Request - %s!\n",
        req);
    return (-1);
      }
#endif              /* WANT */
      phe = htLookup(pCmdData->htCmds, req);
      if (phe == NULL) {
    fprintf(stderr, "shaClientHandler()->Unknown Saved Request - %s!\n",
        req);
    return (-1);
      }
      pCmd = (struct cmCommand *) phe->data;
      (*pCmd->function) (fd, (char *) hReq->arg);
      node = pHost->sendList->head;
      listDeleteThis(pHost->sendList, node);
#ifdef DEBUG
      fprintf(stderr, "shaClientHandler()->acked and deleted %s!\n", req);
#endif              /* DEBUG */
      free(buf);
      free(hReq);
      free(node);
      /* delete req from the queue */
    } else {    /* maybe this is an inbound command! */
      int             retVal;
      retVal = cmNewSearchNExecute(fd, buf, pCmdData->htCmdsIn, arg);
      /*
       * retVal =  cmSearchNExecute(fd,buf, pCmdsIn,
       * pCmdData->nCmdsIn,arg);
       */
#ifdef DEBUG
      fprintf(stderr, "shaClientHandler()->inbound %s!\n", req);
#endif              /* DEBUG */
      free(buf);
      if (retVal == -1) {
    fprintf(stderr, "shaClientHandler()->Unintelligible Response : %s!\n"
        ,buf);
      }
      return -1;
    }
  }

  return 0;
}
```

```
/***********************************************************************
    ***/
/***********************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***********************************************************************
    ***/
/***********************************************************************
    ***/
#include <stdio.h>
#include <errno.h>
#include <poll.h>
#include <sys/time.h>
#ifdef SHASTRA4SUN5
#include <sys/resource.h>
#endif
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <netdb.h>
#include <malloc.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>
#include <shastra/network/serverP.h>
#include <shastra/network/mplexP.h>
#include <shastra/network/mplex.h>
#include <shastra/network/sharedMem.h>
#include <shastra/network/rpc.h>

#include <shastra/utils/dllist.h>

#define MYBUFSIZE  /*32768 32768, 65536 , 102400*/ 131072
#define USE_STREAMS     /* CHECK same flag in mplex, server */
#define DEFAULTMPLEXTIMEOUT 3600000 /* 1hr */
int fDebug = 0;
mplex *mplexTab;
```

```
static struct pollfd  *mplexPollFds;
static struct dllist  *mplexTimerList;
static int iMplexTimeBase;
static int iNChannels = 0;
static int iMplexTimeout = DEFAULTMPLEXTIMEOUT;
static int iMplexPollTimeout;
static int iMplexTotalIdle = 0;
static int mplexMaxChannels = 0;

static Widget wgMplexTop;
static XtAppContext xacMplex;

int (*mplexErrHandler) (Prot1(int));
int (*mplexIdleHandler) (Prot1(char*));
static int mplexDefaultErrHandler(Prot1(int));
static int mplexDefaultIdleHandler(Prot1(char*));
static void mplexDefaultReadHandler(Prot3(char*, int *, unsigned long*));
static void mplexDefaultWriteHandler(Prot3(char*, int *, unsigned long*));
static void mplexWorkTheTimer();

#ifdef SHASTRA4HP
#include <sys/param.h>      /* for HP's which don't have getdtablesize */
int
getdtablesize()
{
  return NOFILE;
}
#endif              /* getdtablesize */

#ifdef SHASTRA4SUN5
int
getdtablesize()
{
  int res;
  rlim_t rlim_cur;
  rlim_t rlim_max;
  struct rlimit rlp;

  res = getrlimit(RLIMIT_NOFILE, &rlp);
  res = (int)rlp.rlim_cur;
  return(res);
}
#endif              /* getdtablesize */

int
mplexInit(wg, xac)
    Widget wg;
    XtAppContext xac;
{
  struct timeval  tp;
  struct timezone tzp;

  wgMplexTop = wg;
  xacMplex = xac;

  if(mplexTab != NULL){
    return;
  }

  gettimeofday(&tp, &tzp);
  mplexMaxChannels = getdtablesize();
#ifdef DEBUG
  fprintf(stderr, "mplexInit()-> max channels = %d\n", mplexMaxChannels);
#endif
  mplexTab = (mplex *) calloc(mplexMaxChannels, sizeof(mplex));
  mplexPollFds = (struct pollfd *) calloc(mplexMaxChannels,
                      sizeof(struct pollfd));
  mplexErrHandler = mplexDefaultErrHandler;
  mplexTimerList = dllistMakeNew();
  iMplexTimeout = DEFAULTMPLEXTIMEOUT;
  iMplexTimeBase = tp.tv_sec;
  iMplexTotalIdle = 0;

  if (xacMplex)
  {
     mplexWorkTheTimer();
  }
  return 0;
}

shaCmdData     *
mplexGetCmdData(fd)
    int             fd;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    return mplexTab[fd].pCmdData;
  } else {
    fprintf(stderr, "mplexGetCmdData()->Bad Channel Number %d\n", fd);
    return NULL;
  }
}

int
mplexSetCmdData(fd, pCmdData)
    int             fd;
    shaCmdData     *pCmdData;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].pCmdData = pCmdData;
    return 1;
  } else {
    fprintf(stderr, "mplexSetCmdData()->Bad Channel Number %d\n", fd);
    return 0;
  }
}
```

```
hostData        *
mplexGetHostData(fd)
    int             fd;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    return mplexTab[fd].pHost;
  } else {
    fprintf(stderr, "mplexGetHostData()->Bad Channel Number %d\n", fd);
    return NULL;
  }
}

int
mplexSetHostData(fd, pHost)
    int             fd;
    hostData       *pHost;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].pHost = pHost;
    return 1;
  } else {
    fprintf(stderr, "mplexSetHostData()->Bad Channel Number %d\n", fd);
    return 0;
  }
}

char           *
mplexGetChannelReadArg(fd)
    int             fd;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    return mplexTab[fd].readArg;
  } else {
    fprintf(stderr, "mplexGetChannelReadArg()->Bad Channel Number %d\n", fd
        );
    return NULL;
  }
}

int
mplexSetChannelReadArg(fd, arg)
    int             fd;
    char           *arg;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].readArg = arg;
    return 1;
  } else {
    fprintf(stderr, "mplexSetChannelReadArg()->Bad Channel Number %d\n", fd
        );
    return 0;
  }
}
```

```
char            *
mplexGetChannelWriteArg(fd)
    int             fd;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    return mplexTab[fd].writeArg;
  } else {
    fprintf(stderr, "mplexGetChannelArg()->Bad Channel Number %d\n", fd);
    return NULL;
  }
}


int
mplexSetChannelWriteArg(fd, arg)
    int             fd;
    char           *arg;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].writeArg = arg;
    return 1;
  } else {
    fprintf(stderr, "mplexSetChannelWriteArg()->Bad Channel Number %d\n",
        fd);
    return 0;
  }
}

static void
mplexDefaultReadHandler(arg, pfd, plId)
     char* arg;
     int *pfd;
     unsigned long* plId;
{
  int i;

  i = *pfd;

  if ((mplexTab[i].fRead) && (mplexTab[i].readHandler != NULL))
  {
    (*mplexTab[i].readHandler) (mplexTab[i].iSocket,mplexTab[i].readArg);
#ifdef USE_STREAMS
    while (mplexTab[i].inStream && (mplexTab[i].inStream->_cnt > 0))
    {
     (*mplexTab[i].readHandler) (mplexTab[i].iSocket,mplexTab[i].readArg);
    }
#endif              /* USE_STREAMS */
  }
}

static void
mplexDefaultWriteHandler(arg, pfd, plId)
```

```
     char* arg;
     int *pfd;
     unsigned long* plId;
{
  int i;

  i = *pfd;

#ifdef SHASTRA4IRIS
  if(mplexTab[i].writeHandler != NULL)
#else
  if((mplexTab[i].fWrite) && (mplexTab[i].writeHandler != NULL))
#endif
  {
    (*mplexTab[i].writeHandler) (mplexTab[i].iSocket, mplexTab[i].writeArg,
                 mplexTab[i].mChanId);
#ifdef USE_STREAMS
    while (mplexTab[i].inStream && (mplexTab[i].inStream->_cnt > 0)) {
      (*mplexTab[i].writeHandler) (mplexTab[i].iSocket,mplexTab[i].writeArg
          ,
                  mplexTab[i].mChanId);
    }

#endif              /* USE_STREAMS */
  }
}

int
mplexRegisterChannel(fd, handler, pCmdData, arg)
    int             fd;
    int             (*handler) ();
    shaCmdData     *pCmdData;
    char           *arg;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse == MPLEX_FREE)){
    memset(&mplexTab[fd], 0, sizeof(mplex));

    mplexTab[fd].pCmdData = pCmdData;
    if (pCmdData != NULL){ /*shaChannel*/
      if(pCmdData->htCmds == NULL) {
    cmInitializeCmdData(pCmdData);
      }

      if (mplexSetFilePtrs(fd) < 0) {
    return -1;
      }
      mplexTab[fd].inBuf = malloc(MYBUFSIZE + 16);
      if (mplexTab[fd].inBuf == NULL) {
    fprintf(stderr, "mplexRegisterChannel()->can't malloc inBuf!\n");
      }
      mplexTab[fd].outBuf = malloc(MYBUFSIZE + 16);
      if (mplexTab[fd].outBuf == NULL) {
    fprintf(stderr, "mplexRegisterChannel()->can't malloc outBuf!\n");
```

```
        }
    /*
      fprintf(stderr, "mplexRegisterChannel(%d)->inBuf = %lx([0]=%c, [%d]=%c, \
      OutBuf=%lx ([0]=%c, [%d]=%c\n",
      fd,
      mplexTab[fd].inBuf, mplexTab[fd].inBuf[0],
      MYBUFSIZE-1, mplexTab[fd].inBuf[MYBUFSIZE-1],
      mplexTab[fd].outBuf, mplexTab[fd].outBuf[0],
      MYBUFSIZE -1, mplexTab[fd].outBuf[MYBUFSIZE-1]);
      */
        if (setvbuf(mplexInStream(fd), mplexTab[fd].inBuf, _IOFBF, MYBUFSIZE)
            ) {
      fprintf(stderr,
          "mplexRegisterChannel()->couldn't setvbuf inBuf!\n");
        }
        if (setvbuf(mplexOutStream(fd), mplexTab[fd].outBuf, _IOFBF,
            MYBUFSIZE)) {
      fprintf(stderr,
          "mplexRegisterChannel()->couldn't setvbuf outBuf!\n");
        }
        mplexTab[fd].pShmInfoIn = shmInfoCreate();
        mplexTab[fd].pShmInfoOut = shmInfoCreate();
      }

      mplexTab[fd].iSocket = fd;
      mplexTab[fd].readHandler = handler;
      mplexTab[fd].fRead = 1;
      mplexTab[fd].readArg = arg;

      mplexTab[fd].fInUse = MPLEX_USE;
      iNChannels++;

      if(xacMplex != NULL)
      {
    #ifdef NVERMINDMENOW
        mplexTab[fd].lChanId =
      XtAppAddInput(xacMplex, fd,
                (XtPointer) XtInputReadMask ,
                mplexDefaultReadHandler, (XtPointer)arg);

        mplexTab[fd].mChanId =
      XtAppAddInput(xacMplex, fd,
                (XtPointer) XtInputWriteMask,
                mplexDefaultWriteHandler, (XtPointer)arg);
    #ifdef NO_SHASTRA4HP
        mplexTab[fd].rChanId =
      XtAppAddInput(xacMplex, fd,
                (XtPointer)  XtInputExceptMask,
                mplexDefaultReadHandler, (XtPointer)arg);
    #endif
    #endif
      }
      else
      {
        mplexTab[fd].lChanId = mplexGetUniqueId();
      }
    } else {
      fprintf(stderr, "mplexRegisterChannel()-> Bad fd = %d\n", fd);
      return -1;
    }
    return 0;
  }

  int
  mplexUnRegisterChannel(fd)
      int             fd;
  {
  /*
    fprintf(stderr, "mplexUnRegisterChannel(%d)\n", fd);
    */
    if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)) {
      iNChannels--;
      if (mplexResetFilePtrs(fd) < 0) {
  /*      mplexTab[fd].fInUse = MPLEX_ERR; */
      }
      if (mplexTab[fd].inBuf) {
        free(mplexTab[fd].inBuf);
      }
      if (mplexTab[fd].outBuf) {
        free(mplexTab[fd].outBuf);
      }
      if (mplexTab[fd].pShmInfoIn) {
        shMemDisconnect(mplexTab[fd].pShmInfoIn);
        free(mplexTab[fd].pShmInfoIn);
      }
      if (mplexTab[fd].pShmInfoOut) {
        shMemDisconnect(mplexTab[fd].pShmInfoOut);
        free(mplexTab[fd].pShmInfoOut);
      }
      if(xacMplex != NULL){
       if (mplexTab[fd].lChanId)
       {
        XtRemoveInput(mplexTab[fd].lChanId);
       }
       if (mplexTab[fd].mChanId)
       {
        XtRemoveInput(mplexTab[fd].mChanId);
       }
  #ifdef NO_SHASTRA4HP
        XtRemoveInput(mplexTab[fd].rChanId);
  #endif
      }
      memset(&mplexTab[fd], 0, sizeof(mplex));
      mplexTab[fd].iSocket = -1;
      shutdown(fd, 2);
      close(fd);
    mplexTab[fd].fInUse = MPLEX_FREE;
  } else {
    return -1;
  }
  return 0;
}

int
mplexMain(flushFunc)
    int             (*flushFunc) ();
{
  int             retval;

  if(xacMplex != NULL){
    XtAppMainLoop(xacMplex);
    return;
  }
  iMplexPollTimeout = iMplexTimeout;
  while (1) {
    retval = mplexPoll(iMplexPollTimeout);
    if (retval == 0) {
      mplexTimeoutHandler();
    } else {
      iMplexTotalIdle = 0;
      mplexTimerTick();
    }
    if (flushFunc != NULL) {
      flushFunc();
    }
  }
  /* NOTREACHED */
}
int
mplexSelect(timeVal)
    int             timeVal;
{
  int             retval;
  int             i;
  int             n;
  int             nDone;

  fd_set          iReadMask, iWriteMask, iExcepnMask;
  struct timeval  timeout;

  FD_ZERO(&iReadMask);
  FD_ZERO(&iWriteMask);
  FD_ZERO(&iExcepnMask);
  for (i = 0, n = 0; (i < mplexMaxChannels) && (n < iNChannels); i++) {
    if (mplexTab[i].fInUse == MPLEX_USE) {
      n++;
      if (mplexTab[i].fWrite) { /* WriteFlag */
    FD_SET(mplexTab[i].iSocket, &iWriteMask);
      }
```

```
        if (mplexTab[i].fRead) {
      FD_SET(mplexTab[i].iSocket, &iReadMask);
        }
      }
    }

    if (fDebug) {
      fprintf(stderr, "before rmask : %ld, wmask : %ld, xmask : %ld\n",
          iReadMask.fds_bits[0],
          iWriteMask.fds_bits[0],
          iExcepnMask.fds_bits[0]);
    }
    if (timeVal > 0) {
      memset((char *) &timeout, 0, sizeof(timeout));
      timeout.tv_sec = timeVal / 1000;
      timeout.tv_usec = (timeVal % 1000) * 1000;
    }
    if ((retval = select(mplexMaxChannels + 1, &iReadMask, &iWriteMask,
                 &iExcepnMask,
                 ((timeVal > 0) ? (&timeout) : NULL))) < 0) {
      extern int      errno;
      if (errno != EINTR) {
        perror("select()");
      }
      return retval;
    }
    if (retval == 0) {        /* timed out */
      return retval;
    } else {
      if (fDebug) {
        fprintf(stderr, "Sel'd %d descriptors\n", retval);
      }
    }
    if (fDebug) {
      fprintf(stderr, "selected rmask : %ld, wmask : %ld, xmask : %ld\n",
          iReadMask.fds_bits[0], iWriteMask.fds_bits[0],
          iExcepnMask.fds_bits[0]);
    }
    nDone = 0;
    for (i = 0, n = 0; (i < mplexMaxChannels) &&
         (n < iNChannels) && (nDone < retval); i++) {
      if (mplexTab[i].fInUse == MPLEX_USE) {
        n++;
        if (mplexTab[i].fWrite && FD_ISSET(mplexTab[i].iSocket, &iWriteMask)
            &&
        (mplexTab[i].writeHandler != NULL)) {
      (*mplexTab[i].writeHandler)(mplexTab[i].iSocket, mplexTab[i].writeArg,
                        mplexTab[i].lChanId);
      nDone++;
        }
        else if (mplexTab[i].fRead && FD_ISSET(mplexTab[i].iSocket, &
            iReadMask)
             && (mplexTab[i].readHandler != NULL)) {
```

```
	(*mplexTab[i].readHandler) (mplexTab[i].iSocket, mplexTab[i].readArg,
				    mplexTab[i].lChanId);
#ifdef USE_STREAMS
	while (mplexTab[i].inStream && (mplexTab[i].inStream->_cnt > 0)) {
/*
  fprintf(stderr,"mplex channel %d->%d\n",i,mplexTab[i].inStream->_cnt);
  */
	  (*mplexTab[i].readHandler)(mplexTab[i].iSocket, mplexTab[i].readArg,
				     mplexTab[i].lChanId);
	}
#endif			/* USE_STREAMS */
	nDone++;
	  }
	}
  }

  return retval;
}

int
mplexPoll(timeout)
	int		timeout;
{
  int		retval;
  int		i;
  unsigned long	n;
  int		nDone;

  for (i = 0, n = 0; (i < mplexMaxChannels) && (n < iNChannels); i++) {
    if (mplexTab[i].fInUse == MPLEX_USE) {
      mplexPollFds[n].fd = i;
      mplexPollFds[n].events = 0;
      mplexPollFds[n].revents = 0;
      if (mplexTab[i].fWrite) { /* WriteFlag */
	mplexPollFds[n].events = POLLOUT;
      }
      if (mplexTab[i].fRead) {
	mplexPollFds[n].events = POLLIN;
      }
      n++;
    }
  }

  if ((retval = poll(mplexPollFds, n, timeout)) < 0) {
    extern int	errno;
    if (errno != EINTR) {
      perror("poll()");
    }
    return retval;
  }
  if (retval == 0) {		/* timed out */
    return retval;
  } else {
      if (fDebug) {
        fprintf(stderr, "Sel'd %d descriptors\n", retval);
      }
    }
    nDone = 0;
    for (n = 0; (n < iNChannels) && (nDone < retval); n++) {
      if (mplexPollFds[n].revents > 0) {
        nDone++;
        i = mplexPollFds[n].fd;
        if ((mplexTab[i].fWrite) && (mplexPollFds[n].revents & POLLOUT) &&
        (mplexTab[i].writeHandler != NULL)) {
      (*mplexTab[i].writeHandler) (mplexTab[i].iSocket, mplexTab[i].writeArg,
                        mplexTab[i].lChanId);
        }
        if ((mplexTab[i].fRead) && (mplexPollFds[n].revents & POLLIN) &&
        (mplexTab[i].readHandler != NULL)) {
      (*mplexTab[i].readHandler) (mplexTab[i].iSocket,mplexTab[i].readArg,
                       mplexTab[i].lChanId);
  #ifdef USE_STREAMS
      while (mplexTab[i].inStream && (mplexTab[i].inStream->_cnt > 0)) {
  /*
    fprintf(stderr,"mplex channel %d->%d\n",i,mplexTab[i].inStream->_cnt);
    */
        (*mplexTab[i].readHandler) (mplexTab[i].iSocket,mplexTab[i].readArg,
                         mplexTab[i].lChanId);
      }
  #endif              /* USE_STREAMS */
        }
      }
    }

    return retval;
  }


  /*--------------------------------------------------------------------------
      --
   * mplexGetFilePtrs(fd,pInStream,pOutStream) -- get file ptrs for the
       channel
   *
   *--------------------------------------------------------------------------
       --
   */

  int
  mplexGetFilePtrs(fd, pInStream, pOutStream)
      int             fd;
      FILE          **pInStream;
      FILE          **pOutStream;
  {
    if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
      if(pInStream){
        *pInStream = mplexInStream(fd);
    }
    if(pOutStream){
      *pOutStream = mplexOutStream(fd);
    }
  } else {
    fprintf(stderr, "mplexGetFilePtrs()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}


/*-------------------------------------------------------------------------
    --
 * mplexSetFilePtrs(fd) -- set file ptrs for the channel
 *
 *-------------------------------------------------------------------------
     --
 */

int
mplexSetFilePtrs(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse == MPLEX_FREE)){
    mplexInStream(fd) = fdopen(fd, "r");
    if (mplexInStream(fd) == NULL) {
      perror("fdopen() In");
      mplexUnRegisterChannel(fd);
      return -1;
    }
    mplexOutStream(fd) = fdopen(fd, "w");
    if (mplexOutStream(fd) == NULL){
      perror("fdopen() Out");
      mplexUnRegisterChannel(fd);
      return -1;
    }
    xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
    xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);

    mplexTab[fd].fInUse = MPLEX_USE;
  } else {
    fprintf(stderr, "mplexSetFilePtrs()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}

/*-------------------------------------------------------------------------
    --
 * mplexResetFilePtrs(fd) -- Reset file ptrs for the channel
 *
 *-------------------------------------------------------------------------
```

```
      --
 */
int
mplexResetFilePtrs(fd)
    int             fd;
{
  if ((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    if(mplexOutStream(fd)){
      xdr_destroy(mplexXDRSEnc(fd));
      fflush(mplexOutStream(fd));
      fclose(mplexOutStream(fd));
      mplexOutStream(fd) = NULL;
    }
    if(mplexInStream(fd)){
      xdr_destroy(mplexXDRSDec(fd));
      fflush(mplexInStream(fd));
      fclose(mplexInStream(fd));
      mplexInStream(fd) = NULL;
    }
    mplexTab[fd].fInUse = MPLEX_FREE;
  } else {
    fprintf(stderr, "mplexResetFilePtrs()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}

/*--------------------------------------------------------------------------
    --
 * mplexSetXDRFlag(fd) -- set xdr flag for channel
 *
 *--------------------------------------------------------------------------
      --
 */
int
mplexSetXDRFlag(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fXDR = 1;
  } else {
    fprintf(stderr, "mplexSetXDRFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}


/*--------------------------------------------------------------------------
    --
 * mplexResetXDRFlag(fd) -- Reset file ptrs for the channel
 *
 *--------------------------------------------------------------------------
```

```
      --
 */
int
mplexResetXDRFlag(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fXDR = 0;
  } else {
    fprintf(stderr, "mplexResetXDRFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}

int
mplexGetMaxChannels()
{
  if (!mplexMaxChannels) {
    mplexMaxChannels = getdtablesize();
  }
  return mplexMaxChannels;
}



int
mplexRegisterErrHandler(handler)
    int             (*handler) ();
{
  if (handler != NULL) {
    mplexErrHandler = handler;
  }
}

static int
mplexDefaultErrHandler(fd)
    int             fd;
{
  mplexUnRegisterChannel(fd);
}

/*---------------------------------------------------------------------------
    --
 * mplexSetReadHandler(fd,handler,arg) -- set read handler
 *
 *--------------------------------------------------------------------------
      --
 */
int
mplexSetReadHandler(fd, handler, arg)
    int             fd;
    int             (*handler) ();
    char           *arg;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].readHandler = handler;
    mplexTab[fd].readArg = arg;
  } else {
    fprintf(stderr, "mplexSetReadHandler()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}


/*------------------------------------------------------------------------
    --
 * mplexSetReadFlag(fd) -- set write flag for channel
 *
 *------------------------------------------------------------------------
     --
 */

int
mplexSetReadFlag(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fRead = 1;
  } else {
    fprintf(stderr, "mplexSetReadFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}



/*------------------------------------------------------------------------
    --
 * mplexResetReadFlag(fd) -- Reset writeFlag for the channel
 *
 *------------------------------------------------------------------------
     --
 */

int
mplexResetReadFlag(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fRead = 0;
  } else {
    fprintf(stderr, "mplexResetReadFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}

/*----------------------------------------------------------------------------
    --
 * mplexSetWriteHandler(fd,handler,arg) -- set write handler
 *
 *----------------------------------------------------------------------------
     --
 */

int
mplexSetWriteHandler(fd, handler, arg)
     int              fd;
     int              (*handler) ();
     char            *arg;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].writeHandler = handler;
    mplexTab[fd].writeArg = arg;
  } else {
    fprintf(stderr, "mplexSetWriteHandler()->Bad Channel Number %d\n",
        fd);
    return -1;
  }
  return 0;
}

/*----------------------------------------------------------------------------
    --
 * mplexSetWriteFlag(fd) -- set write flag for channel
 *
 *----------------------------------------------------------------------------
     --
 */

int
mplexSetWriteFlag(fd)
     int              fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fWrite = 1;
  } else {
    fprintf(stderr, "mplexSetWriteFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}



/*----------------------------------------------------------------------------
```

```
    --
 * mplexResetWriteFlag(fd) -- Reset writeFlag for the channel
 *
 *-----------------------------------------------------------------------
     --
 */

int
mplexResetWriteFlag(fd)
    int             fd;
{
  if((fd >= 0) && (mplexTab[fd].fInUse != MPLEX_FREE)){
    mplexTab[fd].fWrite = 0;
  } else {
    fprintf(stderr, "mplexResetWriteFlag()->Bad Channel Number %d\n", fd);
    return -1;
  }
  return 0;
}

unsigned long
mplexRegisterIdler(handler, arg)
     int (*handler)();
     char *arg;
{
  if (handler != NULL) {
    if(xacMplex != NULL){
      return XtAppAddWorkProc(xacMplex, (char (*)())handler, (XtPointer)arg
          );
    }
    else{
      mplexIdleHandler = handler;
    }
  }
}

int
mplexUnRegisterIdler(lWPId)
     unsigned long lWPId;

{
  if(xacMplex != NULL){
    XtRemoveWorkProc(lWPId);
  }
  else{
    mplexIdleHandler = NULL;
  }
}

static int
mplexDefaultIdleHandler(arg)
     char *arg;
{
  fprintf(stderr, "mplexDefaultIdleHandler()-> called\n");
}

/*------------------------------------------------------------------------
    --
 * mplexSetTimeout(iTime) -- set timeout value for mplex..
 * process will exit after this
 *
 *------------------------------------------------------------------------
     --
 */

int
mplexSetTimeout(iTime)
    int             iTime;
{
  iMplexTimeout = iTime;
}
/*------------------------------------------------------------------------
    --
 * mplexGetTimeout(iTime) -- get timeout value for mplex
 * process will exit after this
 *
 *------------------------------------------------------------------------
     --
 */

int
mplexGetTimeout(iTime)
    int             iTime;
{
  return iMplexTimeout;
}

int
mplexRegisterTimer(iDelay, timerHandler, timerArg)
     unsigned long iDelay;
     void (*timerHandler) (Prot2(char*, unsigned long*));
     char *timerArg;
{
  struct dllist_node *tmpNode, *node;
  mplexTimerData *timerData, *tData;
  struct timeval  tp;
  struct timezone tzp;
  int             msecTime, sepTime;

  if(xacMplex != NULL){
    return XtAppAddTimeOut(xacMplex, iDelay, timerHandler,
               (XtPointer)timerArg);
  }
  gettimeofday(&tp, &tzp);
  timerData = (mplexTimerData *) malloc(sizeof(mplexTimerData));
  msecTime = tp.tv_usec / 1000 + (tp.tv_sec - iMplexTimeBase) * 1000;
  timerData->iTimerId = tp.tv_usec + tp.tv_sec;
  timerData->timerHandler = timerHandler;
  timerData->timerArg = timerArg;
  timerData->iTimeout = iDelay + msecTime;
  timerData->iDeltaTime = iDelay;

  tmpNode = dllistMakeNewNode();
  tmpNode->data = (char *) timerData;

  if (mplexTimerList->head == NULL) {
    dllistInsertAtTail(mplexTimerList, tmpNode);
  } else {
    for (node = mplexTimerList->head; node != NULL; node = node->next) {
      tData = (mplexTimerData *) node->data;
      if (tData->iTimeout > timerData->iTimeout) {
    break;
      }
    }
    if (node == NULL) {
      dllistInsertAtTail(mplexTimerList, tmpNode);
      timerData->iDeltaTime = timerData->iTimeout - tData->iTimeout;
      if (timerData->iDeltaTime > iDelay) {
    timerData->iDeltaTime = iDelay;
      }
      timerData->iTimeout = tData->iTimeout + timerData->iDeltaTime;
    } else {
      dllistInsertBefore(mplexTimerList, node, tmpNode);
      sepTime = tData->iTimeout - timerData->iTimeout;
      timerData->iDeltaTime = tData->iDeltaTime - sepTime;
      tData->iDeltaTime = sepTime;
    }
  }
  if(tmpNode == mplexTimerList->head){
    iMplexPollTimeout = timerData->iDeltaTime;
  }
  return timerData->iTimerId;
}

int
mplexHandleTimer()
{
  struct dllist_node *tmpNode, *node;
  mplexTimerData *tData;

  for (node = mplexTimerList->head; node != NULL;) {
    tData = (mplexTimerData *) node->data;
    if (tData->iDeltaTime > 0) {
      break;
    }
    tmpNode = node->next;
    /* handled, remove.. else handler may try to unregister.. */
    dllistDeleteThis(mplexTimerList, node);
    /* expired, execute.. this might add more nodes */
    (*tData->timerHandler) (tData->timerArg, tData->iTimerId);
    free(node->data);
    free(node);
    node = tmpNode;

#ifdef WANTTHISADJUST
    mplexTimerData *t2Data;
    /* this adjusts for negative time..danger of backlog */
    if ((tData->iDeltaTime < 0) && (tmpNode != NULL)) {
      t2Data = (mplexTimerData *) tmpNode->data;
      t2Data->iDeltaTime += tData->iDeltaTime;
    }
#endif              /* WANT */
  }
  if (mplexTimerList->head != NULL) {
    tData = (mplexTimerData *) mplexTimerList->head->data;
    iMplexPollTimeout = tData->iDeltaTime;
  } else {
    iMplexPollTimeout = iMplexTimeout;
  }
}

int
mplexTimerTick()
{
  mplexTimerData *tData;
  struct timeval  tp;
  struct timezone tzp;
  int             msecTime;

  if (mplexTimerList->head != NULL) {
#ifdef DEBUG
    showTimer();
#endif              /* DEBUG */
    gettimeofday(&tp, &tzp);
    tData = (mplexTimerData *) mplexTimerList->head->data;
    msecTime = tp.tv_usec / 1000 + (tp.tv_sec - iMplexTimeBase) * 1000;
    tData->iDeltaTime = tData->iTimeout - msecTime;
    if (tData->iDeltaTime <= 0) {
      mplexHandleTimer();
    } else {
      iMplexPollTimeout = tData->iDeltaTime;
    }
#ifdef DEBUG
    showTimer();
#endif              /* DEBUG */
  }
}

int
mplexTimeoutHandler()
{
  if (iMplexPollTimeout == iMplexTimeout) {
      iMplexTotalIdle += iMplexTimeout;
      if (iMplexTotalIdle >= (DEFAULTMPLEXTIMEOUT * 10)) {
        fprintf(stderr, "mplexTimeoutHandler()->timed out and died!\n");
        exit(-1);
      }
    }
    if (mplexTimerList->head == NULL) {
      if (mplexIdleHandler) {
        (*mplexIdleHandler) (0);
      }
      iMplexPollTimeout = iMplexTimeout;
    } else {
      mplexTimerTick();
    }
  }

  int
  showTimer()
  {
    mplexTimerData *tData;
    struct dllist_node *node;
    int             i;

    for (node = mplexTimerList->head, i = 0; node != NULL;
         node = node->next, i++) {
      tData = (mplexTimerData *) node->data;
      fprintf(stderr, "[%d]--%d (%d)\n", i, tData->iDeltaTime, tData->
          iTimeout);
    }
  }


  int
  mplexUnRegisterTimer(iTimerId)
       unsigned long iTimerId;
  {
    struct dllist_node *node;
    mplexTimerData *t2Data, *tData;

    if(xacMplex != NULL){
      XtRemoveTimeOut(iTimerId);
      return;
    }
    for (node = mplexTimerList->head; node != NULL; node = node->next) {
      tData = (mplexTimerData *) node->data;
      if (tData->iTimerId == iTimerId) {
        if (node->next != NULL) {
      t2Data = (mplexTimerData *) node->next->data;
      t2Data->iDeltaTime += tData->iDeltaTime;
        }
        dllistDeleteThis(mplexTimerList, node);
        free(node->data);
        free(node);
        break;
      }
    }
}

unsigned long
mplexGetUniqueId()
{
  struct timeval  tp;
  struct timezone tzp;
  int             id;

  gettimeofday(&tp, &tzp);
  id = tp.tv_usec + tp.tv_sec;  /* unique enough */

  return id;
}



static void mplexRegisterAllInputFuncs()
{
  int i;
  for (i = 0; i < mplexMaxChannels; i++)
  {
    if (mplexTab[i].fInUse == MPLEX_USE)
    {
     if (mplexTab[i].fRead)
     {
      mplexTab[i].lChanId =
    XtAppAddInput(xacMplex, i,
              (XtPointer) XtInputReadMask ,
              mplexDefaultReadHandler, (XtPointer)mplexTab[i].readArg);
     }

     if (mplexTab[i].fWrite)
     {
       fprintf(stderr, "Somone to write to!\n");
      mplexTab[i].mChanId =
    XtAppAddInput(xacMplex, i,
              (XtPointer) XtInputWriteMask,
              mplexDefaultWriteHandler, (XtPointer)mplexTab[i].readArg);
      }
#ifdef NO_SHASTRA4HP
       fprintf(stderr, "Somone to except  to!\n");
      mplexTab[i].rChanId =
    XtAppAddInput(xacMplex, i,
              (XtPointer)  XtInputExceptMask,
              mplexDefaultReadHandler, (XtPointer)mplexTab[i].readArg);
#endif
     }
   }
}
```

```
static void mplexUnRegisterAllInputFuncs()
{
  int i;
  for (i = 0; i < mplexMaxChannels; i++)
  {
    if (mplexTab[i].fInUse == MPLEX_USE)
    {
     if ((mplexTab[i].fRead) && (mplexTab[i].lChanId))
     {
      XtRemoveInput(mplexTab[i].lChanId);
      mplexTab[i].lChanId = 0;
     }
     if ((mplexTab[i].fWrite) && (mplexTab[i].mChanId))
     {
       fprintf(stderr, "no one to write to!\n");
      XtRemoveInput(mplexTab[i].mChanId);
      mplexTab[i].mChanId = 0;
     }
#ifdef NO_SHASTRA4HP
       fprintf(stderr, "no one to except to!\n");
      XtRemoveInput(mplexTab[i].rChanId);
#endif
     }
   }
}

static void mplexWorkTheTimer()
{
    static int flag = 1;
    if (flag)
    {
       mplexRegisterAllInputFuncs();
       flag = 0;
    }
    else
    {
       mplexUnRegisterAllInputFuncs();
       flag = 1;
    }
    XtAppAddTimeOut(xacMplex, 50, mplexWorkTheTimer,
              (XtPointer)NULL);
}
```

```
/***************************************************************************
    ***/
/***************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/***************************************************************************
    ***/
/***************************************************************************
    ***/
#include <stdio.h>
#include <errno.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define STANDALONEnn
#define TEST_OnFD

/*---------------------------------------------------------------------------
    --
 * pipeSlaveOnFD -- create a pipe slave on a given a file descriptor
 *
 * Arguments are
 * one file descriptor used for reading and writing to the Slave process.
 * argv initialized for the slave (null terminated)
 *
 * The routine forks and executes a Slave process and sets up
 * the descriptors so it is talking via stdio to the Slave process.
 *
 * returns -1 on error
 *--------------------------------------------------------------------------
     --
 */
int
pipeSlaveOnFD(fdIO, argv)
    int             fdIO;
    char          **argv;
{
    int            e;

#ifdef SHASTRA4SUN4
    if ((e = vfork()) == 0) {
#else              /* SHASTRA4SUN4 -- SGI */
    if ((e = fork()) == 0) {
#endif             /* SHASTRA4SUN4 */
        if (dup2(fdIO, 0) == -1 || dup2(fdIO, 1) == -1) {
            perror("dup2()");
            return -1;
        }
        /* now exec the Slave program */
        if (execv(argv[0], argv) == -1) {
            perror("execv()");
            return -1;
        }
    } else if (e == -1) {
        perror("fork()");
        return -1;
    }
    wait3(NULL, WNOHANG, NULL);
    return e;       /* good return */
}


/*---------------------------------------------------------------------------
    --
 * pipeSlave -- create a pipe slave
 *
 * Arguments are
 * one file descriptor pointer which returns a descriptor to be
 *    used for reading and writing to the Slave process.
 * argv initialized for the slave(null terminated)
 *
 * The routine forks and executes a Slave process and sets up
 * the descriptors so it is talking via stdio to the Slave process.
 *
 * returns -1 on error
 *---------------------------------------------------------------------------
     --
 */

int
pipeSlave(pFdIO, argv)
    int           *pFdIO;
    char          **argv;
{
    int            sockPair[2];
    int            e;

    if (socketpair(AF_UNIX, SOCK_STREAM, 0, sockPair) == -1) {
        perror("socketpair():");
        return -1;
    }
    /* set up a two-way pipe */
#ifdef SHASTRA4SUN4
    if ((e = vfork()) == 0) {
#else              /* SHASTRA4SUN4 -- SGI */
    if ((e = fork()) == 0) {
#endif             /* SHASTRA4SUN4 */
        /* in child */
        if (close(sockPair[0]) == -1) {
            perror("close():");
            return -1;
        }
        if (dup2(sockPair[1], 0) == -1) {
            perror("dup2():0");
            return -1;
        }
        if (dup2(sockPair[1], 1) == -1) {
            perror("dup2():1");
            return -1;
        }
        if (execv(argv[0], argv) == -1) {
            perror("execv():");
            return -1;
        }
        exit(0);
    } else if (e == -1) {
        perror("fork()");
        return -1;
    }
    /* in parent */
    if (close(sockPair[1]) == -1) {
        perror("close():");
        return -1;
    }
    *pFdIO = sockPair[0];
    return e;      /* good return to main process */

}

/*--------------------------------------------------------------------------
    --
 * remotePipeSlaveOnFD -- create a pipe slave on a given a file descriptor
 *                       on a remote host
 *
 * Arguments are
 * one file descriptor used for reading and writing to the Slave process.
 * host name of remote host
 * argv initialized for the slave (null terminated)
 *
 * The routine forks and executes a Slave process and sets up
 * the descriptors so it is talking via stdio to the Slave process.
 *
 * returns -1 on error
 *--------------------------------------------------------------------------
     --
 */
int
remotePipeSlaveOnFD(fdIO, hostname, argv)
    int             fdIO;
    char            *hostname;
    char          **argv;
{
    int             e;

#ifdef SHASTRA4SUN4
    if ((e = vfork()) == 0) {
#else               /* SHASTRA4SUN4 -- SGI */
    if ((e = fork()) == 0) {
#endif              /* SHASTRA4SUN4 */
        if (dup2(fdIO, 0) == -1 || dup2(fdIO, 1) == -1) {
            perror("dup2()");
            return -1;
        }
        /* now exec the Slave program */
/*ACTUALLY -- create new argv, with /usr/ucb/rsh hostname etc */
        if (execv(argv[0], argv) == -1) {
            perror("execv()");
            return -1;
        }
    } else if (e == -1) {
        perror("fork()");
        return -1;
    }
    return e;       /* good return */
}


/*--------------------------------------------------------------------------
    --
 * remotePipeSlave -- create a pipe slave
 *
 * Arguments are
 * one file descriptor pointer which returns a descriptor to be
 *     used for reading and writing to the Slave process.
 * hostname of remote host
 * argv initialized for the slave(null terminated)
 *
 * The routine forks and executes a Slave process and sets up
 * the descriptors so it is talking via stdio to the Slave process.
 *
 * returns -1 on error
 *--------------------------------------------------------------------------
     --
 */
```

```
int
remotePipeSlave(pFdIO, hostname, argv)
    int             *pFdIO;
    char            *hostname;
    char           **argv;
{
    int              sockPair[2];
    int              e;
    char           **newArgv;

    if (socketpair(AF_UNIX, SOCK_STREAM, 0, sockPair) == -1) {
        perror("socketpair():");
        return -1;
    }
    /* set up a two-way pipe */
#ifdef SHASTRA4SUN4
    if ((e = vfork()) == 0) {
#else                /* SHASTRA4SUN4 -- SGI */
    if ((e = fork()) == 0) {
#endif               /* SHASTRA4SUN4 */
        /* in child */
        if (close(sockPair[0]) == -1) {
            perror("close():");
            return -1;
        }
        if (dup2(sockPair[1], 0) == -1) {
            perror("dup2():0");
            return -1;
        }
        if (dup2(sockPair[1], 1) == -1) {
            perror("dup2():1");
            return -1;
        }
/*ACTUALLY -- create new argv, with /usr/ucb/rsh hostname etc */
    /*now exec an rsh host cmd*/
    newArgv = (char**)malloc(sizeof(char*)*4);
    newArgv[0] = strdup("/usr/ucb/rsh");
    newArgv[1] = strdup(hostname);
    newArgv[2] = strdup(argv[0]);
    newArgv[3] = NULL;
        if (execv(newArgv[0], newArgv) == -1) {
            perror("execv():");
            return -1;
        }
        exit(0);
    } else if (e == -1) {
        perror("fork()");
        return -1;
    }
    /* in parent */
    if (close(sockPair[1]) == -1) {
        perror("close():");
        return -1;
    }
    *pFdIO = sockPair[0];
    return e;       /* good return to main process */

}


#ifdef STANDALONE
main(argc, argv)
    int             argc;
    char          **argv;
{
    int             fd;
    static char    *argvSlave[] = {
        "/usr/bin/tr",
        "[a-z]",
        "[A-Z]",
        NULL
    };
#ifdef TEST_OnFD
    fd = 1;         /* stdout descriptor */
    if (pipeSlaveOnFD(fd, argv) == -1) {
        fprintf(stderr, "pipeSlaveOnFD()->failed!\n");
        return;
    }
#else               /* TEST_OnFD-- no FD */
    if (pipeSlave(&fd, argv) == -1) {
        fprintf(stderr, "pipeSlave()->failed!\n");
        return;
    }
#endif              /* TEST_OnFD */
    {
        FILE           *fp;
        int             i;
        char           *str;
        char            sb[1024];

        fp = fdopen(fd, "w");
        for (i = 0; i < 10; i++) {
            fprintf(fp, "abcdefghijklmnopqrstuvwxyz\n");
        }
        fclose(fp);
        fp = fdopen(fd, "r");
        for (i = 0; i < 10; i++) {
            str = fgets(sb, 1024, fp);
            fprintf(stdout, "%s\n", str);
        }
        fclose(fp);
    }
}
#endif              /* STANDALONE */
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <errno.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <netdb.h>

#include <shastra/shastra.h>

#include <shastra/utils/hash.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>
#include <shastra/network/serverP.h>
#include <shastra/network/serverPorts.h>
#include <shastra/network/mplexP.h>
#include <shastra/network/mplex.h>

#include <shastra/datacomm/shastraDataH.h>

#define DEBUGxx
#define USEXDRnn
#define USE_STREAMS     /* CHECK same flag in mplex, server */

char *readString(Prot1(int));
char **readStrings(Prot1(int));
int cmNewHandleCmdConnection(Prot3(int, hashTable *, char *));
int cmNewSearchNExecute(Prot4(int, char *, hashTable *, char *));

static struct sockaddr_in saInServer;

/*
 * Function
 */
unsigned long
hostName2IPAddress(sName)
     char          *sName;
{
  struct hostent *pHostEnt;
  if (sName == NULL || (pHostEnt = gethostbyname(sName)) == NULL) {
    return 0;
  } else
  {
    unsigned int temp;
    memcpy(&temp, &pHostEnt->h_addr_list[0][0], 4);
    return ntohl(temp);
  }
}

int
cmCloseSocket(iSocket)
    int             iSocket;
{
  if (shutdown(iSocket, 2) != 0) {
    perror("shutdown()");
    return -1;
  }
  if (close(iSocket) != 0) {
    perror("close()");
    return -1;
  }
  return 0;
}

int
cmPrintErr(sMessage)
    char           *sMessage;
{
#ifdef DEBUG
    perror(sMessage);
#endif
    sMessage = NULL;
#ifdef ERR_EXIT
    exit(-1);
#else
    return -1;
#endif
}
```

```
/*
 * Function
 */
int
cmOpenServerSocket(sService, iPort, pCmdData, pSocket, argRead)
    char            *sService;
    int              iPort;
    shaCmdData      *pCmdData;
    int             *pSocket;
    char            *argRead;
{
  int              length;

  struct servent *pServEnt;
  struct protoent *pProtoEnt;
  struct linger   soLinger;
  int              iOption;
  int              fNonStdPort = 0;
  int              iSocket;

  pProtoEnt = getprotobyname("tcp");
  if(iPort > 0){
    fNonStdPort = 1;
  }
  else if (((pServEnt = getservbyname(sService, "tcp")) == NULL) ||
      (pServEnt->s_port == 0)) {
    fNonStdPort = 1;
    iPort = getServerPort(sService);
  }
  if((iSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    perror("socket()");
    return -1;
  }
#ifdef DEBUG
  fprintf(stderr, "got socket descriptor %d\n", iSocket);
#endif              /* DEBUG */
  soLinger.l_onoff = 0;
  soLinger.l_linger = 5;    /* seconds */
  if (setsockopt(iSocket, SOL_SOCKET, SO_LINGER, &soLinger,
        sizeof(struct linger)) == -1) {
    perror("setsockopt() SOL_SOCKET, SO_LINGER");
    close(iSocket);
    return -1;
  }
  saInServer.sin_family = AF_INET;
  saInServer.sin_addr.s_addr = INADDR_ANY;
  if (fNonStdPort) {
    saInServer.sin_port = htons(iPort);
  } else {
    saInServer.sin_port = pServEnt->s_port;
    iPort = ntohs(pServEnt->s_port);
  }
  if(bind(iSocket, &saInServer, sizeof(saInServer)) != 0){
    perror("bind()");
    close(iSocket);
    return -1;
  }
  length = sizeof(saInServer);
  if (getsockname(iSocket, &saInServer, &length)) {
    perror("getsockname()");
    close(iSocket);
    return -1;
  }
  iPort = ntohs(saInServer.sin_port);
#ifdef DEBUG
  fprintf(stderr, "Tcp Socket has port #%d\n", iPort);
#endif              /* DEBUG */
  if (listen(iSocket, 5) != 0) {
    perror("listen()");
    close(iSocket);
    return -1;
  }
  iOption = 1;
  if (setsockopt(iSocket, SOL_SOCKET, SO_REUSEADDR,
         &iOption, sizeof(iOption)) == -1) {
    perror("setsockopt() SOL_SOCKET, SO_REUSEADDR");
    close(iSocket);
    return -1;
  }
  if (mplexRegisterChannel(iSocket, cmServerAcceptHandler,
               pCmdData, argRead) == -1) {
    close(iSocket);
    return -1;
  };
  *pSocket = iSocket;
  return iPort;
}


int
cmClientConnect2Server(sHost, sService, iPortSvc, pSocket)
    char           *sHost;
    char           *sService;
    int             iPortSvc;
    int            *pSocket;
{
  struct protoent *pProtoEnt;
  struct sockaddr_in saInServer;
  struct hostent *pHostEnt;
  struct servent *pServEnt;
  struct linger   soLinger;
  int             fNonStdPort = 0;
  int             iPort = 0;

  pProtoEnt = getprotobyname("tcp");
  if((*pSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0){
      perror("socket()");
      return -1;
    }
    soLinger.l_onoff = 0;
    soLinger.l_linger = 5;    /* number of seconds */
    if (setsockopt(*pSocket, SOL_SOCKET, SO_LINGER, &soLinger,
           sizeof(struct linger)) == -1) {
      perror("setsockopt() SOL_SOCKET, SO_LINGER");
      return -1;
    }
  #ifdef DEBUG
    fprintf(stderr, "Got socket descr %d for s1\n", *pSocket);
  #endif              /* DEBUG */
    if((pHostEnt = gethostbyname(sHost)) == NULL){
      fprintf(stderr, "Unknown host %s\n", sHost);
      close(*pSocket);
      return (-1);
    }
    if (iPortSvc != 0) {
      fNonStdPort = 1;
      iPort = iPortSvc;
    } else if ((pServEnt = getservbyname(sService, "tcp")) == NULL) {
      fNonStdPort = 1;
      if ((iPort = getServerPort(sService))) {
      } else {
        iPort = iPortSvc;
      }
    }
    memcpy((char*)&saInServer.sin_addr, pHostEnt->h_addr, pHostEnt->h_length)
        ;
    saInServer.sin_family = AF_INET;
    if (fNonStdPort) {
      saInServer.sin_port = htons(iPort);
    } else {
      saInServer.sin_port = pServEnt->s_port;
      iPort = ntohs(pServEnt->s_port);
    }
    if (connect(*pSocket, &saInServer, sizeof(saInServer)) < 0) {
      perror("connect()");
      close(*pSocket);
      return -1;
    }
  #ifdef WANTTHIS
    struct sockaddr_in saInClient;
    int             length = 0;
    length = sizeof(saInClient);
    if (getpeername(*pSocket, &saInClient, &length) < 0) {
      perror("getpeername()");
    } else {
      fprintf(stderr, "ServerPort = %d (len %d)\n",
          ntohs(saInClient.sin_port), length);
    }
    if (getsockname(*pSocket, &saInClient, &length) < 0) {
```

```
    perror("getpeername()");
  } else {
    fprintf(stderr,"ClientPort = %d (len %d)\n",
        ntohs(saInClient.sin_port), length);
  }
#endif              /* WANTTHIS */
  return iPort;
}

int
cmServerAcceptHandler(iSock, argDummy)
    int             iSock;
    char           *argDummy;
{
  int             length;
  int             iSockNew;
  int             retVal;
  int             (*fnConnect) ();

  length = sizeof(saInServer);
  if ((iSockNew = accept(iSock, &saInServer, &length)) < 0) {
    perror("accept()");
    return -1;
  }
#ifdef DEBUG
  fprintf(stderr, "socket descriptor %d for client connection\n", iSockNew)
      ;
#endif              /* DEBUG */
  argDummy = NULL;
  retVal = mplexRegisterChannel(iSockNew, cmHandleServerConnection,
                 mplexTab[iSock].pCmdData, NULL);
  if(retVal == -1){
    close(iSockNew);
    return(-1);
  }
  /* if there is a connect-func, call it */
/*CHECK*/
  if (mplexTab[iSock].readArg != NULL) {
    fnConnect = (int (*) ()) mplexTab[iSock].readArg;
    (*fnConnect) (iSockNew);
  }
  return retVal;
}

int
cmHandleServerConnection(iSock, argDummy)
    int             iSock;
    char           *argDummy;
{
  return cmNewHandleCmdConnection(iSock, mplexTab[iSock].pCmdData->htCmds,
                argDummy);
  /*
   * return cmHandleCmdConnection(iSock,
     * mplexTab[iSock].pCmdData->pCmdTab,
     * mplexTab[iSock].pCmdData->nCmds,argDummy);
     */
}

int
cmInitializeCmdData(pCmdData)
    shaCmdData     *pCmdData;
{
  int             i;
  /* put entries into the hash table */
  if (pCmdData->htCmds == NULL) {
    pCmdData->htCmds = htMakeNew(CMHASHTABLESIZE, 0 /* arbitsize */ );
  }
  for (i = 0; i < pCmdData->nCmds; i++) {
    if (pCmdData->pCmdTab[i].command == NULL) {
      fprintf(stderr, "cmInitializeCmdData()->null command!\n");
    }
    htInstallSymbol(pCmdData->htCmds,
            pCmdData->pCmdTab[i].command,
            (char *) &pCmdData->pCmdTab[i]);
  }
  /*
   * htDump(pCmdData->htCmds,0); htDump(pCmdData->htCmds,1);
   */

  if (pCmdData->htCmdsIn == NULL) {
    pCmdData->htCmdsIn = htMakeNew(CMHASHTABLESIZE, 0 /* arbitsize */ );
  }
  for (i = 0; i < pCmdData->nCmdsIn; i++) {
    htInstallSymbol(pCmdData->htCmdsIn,
            pCmdData->pCmdTabIn[i].command,
            (char *) &pCmdData->pCmdTabIn[i]);
  }
  /*
   * htDump(pCmdData->htCmdsIn,0); htDump(pCmdData->htCmdsIn,1);
   */

}

/*
 * func() -- destructively add cmds to old shaCmdData
 */
int
cmJoinCmdData(pCmdDataOld, pCmdDataAdd)
    shaCmdData     *pCmdDataOld;
    shaCmdData     *pCmdDataAdd;
{
  cmCommand      *pCmdTab;
  cmCommand      *pCmdTabIn;
  int             i;
  int             iNext = 0;
```

```
    pCmdTab = (cmCommand *) malloc(sizeof(cmCommand) *
                   (pCmdDataOld->nCmds + pCmdDataAdd->nCmds));
    if (pCmdDataOld->nCmds > 0) {
      memcpy(&pCmdTab[0],pCmdDataOld->pCmdTab,
         sizeof(cmCommand) * pCmdDataOld->nCmds);
      iNext = pCmdDataOld->nCmds;
    }
    if (pCmdDataAdd->nCmds > 0) {
      memcpy(&pCmdTab[iNext],pCmdDataAdd->pCmdTab,
         sizeof(cmCommand) * pCmdDataAdd->nCmds);
    }
    pCmdDataOld->pCmdTab = pCmdTab;
    pCmdDataOld->nCmds = pCmdDataOld->nCmds + pCmdDataAdd->nCmds;
    if (pCmdDataOld->htCmds == NULL) {
      pCmdDataOld->htCmds = htMakeNew(CMHASHTABLESIZE, 0 /* arbitsize */ );
      iNext = 0;
    }
    for (i = iNext; i < pCmdDataOld->nCmds; i++) {
      htInstallSymbol(pCmdDataOld->htCmds,
              pCmdDataOld->pCmdTab[i].command,
              (char *) &pCmdDataOld->pCmdTab[i]);
    }

    pCmdTabIn = (cmCommand *) malloc(sizeof(cmCommand) *
                     (pCmdDataOld->nCmdsIn + pCmdDataAdd->nCmdsIn));
    if (pCmdDataOld->nCmdsIn > 0) {
      memcpy(&pCmdTabIn[0],pCmdDataOld->pCmdTabIn,
         sizeof(cmCommand) * pCmdDataOld->nCmdsIn);
      iNext = pCmdDataOld->nCmdsIn;
    }
    if (pCmdDataAdd->nCmdsIn > 0) {
      memcpy(&pCmdTabIn[iNext],pCmdDataAdd->pCmdTabIn,
         sizeof(cmCommand) * pCmdDataAdd->nCmdsIn);
    }
    pCmdDataOld->pCmdTabIn = pCmdTabIn;
    pCmdDataOld->nCmdsIn = pCmdDataOld->nCmdsIn + pCmdDataAdd->nCmdsIn;
    if (pCmdDataOld->htCmdsIn == NULL) {
      pCmdDataOld->htCmdsIn = htMakeNew(CMHASHTABLESIZE, 0 /* arbitsize */ );
      iNext = 0;
    }
    /* put entries into the hash table */
    for (i = iNext; i < pCmdDataOld->nCmdsIn; i++) {
      htInstallSymbol(pCmdDataOld->htCmdsIn,
              pCmdDataOld->pCmdTabIn[i].command,
              (char *) &pCmdDataOld->pCmdTabIn[i]);
    }
    return 0;
  }


  int
  cmHandleCmdConnection(iSocket, pCmdTab, nCmds, argDummy)
      int             iSocket;
    cmCommand      *pCmdTab;
    int             nCmds;
    char           *argDummy;
{
  char           *sBuf;
  int             retVal;

  sBuf = cmReceiveString(iSocket);
  if (sBuf == NULL) {
    return (*mplexErrHandler) (iSocket);
  }
  retVal = cmSearchNExecute(iSocket, sBuf, pCmdTab, nCmds, argDummy);
  free(sBuf);
  return retVal;
}

int
cmSearchNExecute(iSocket, sBuf, pCmdTab, nCmds, argDummy)
    int             iSocket;
    char           *sBuf;
    cmCommand      *pCmdTab;
    int             nCmds;
    char           *argDummy;
{
  int             i;
  int             fFound = 0;

  if (sBuf == NULL) {
    return 0;
  }
  for (i = 0; (i < nCmds) && !fFound; i++) {
    if (strncmp(pCmdTab[i].command, sBuf,
        strlen(pCmdTab[i].command)) == 0) {
      fFound = 1;
#ifdef DEBUG
      fprintf(stderr, "%s\n", pCmdTab[i].helpmsg);
#endif              /* DEBUG */
      (*pCmdTab[i].function) (iSocket, argDummy);
      break;
    }
  }
  if (!fFound) {
    fprintf(stderr, "cmSearchNExecute()- Command not found -> %s\n",
        sBuf);
    return (-1);
  }
  return 0;
}

int
cmNewHandleCmdConnection(iSocket, phtCmds, argDummy)
    int             iSocket;
    hashTable      *phtCmds;
    char            *argDummy;
{
  char            *sBuf;
  int              retVal;

  sBuf = cmReceiveString(iSocket);
  if (sBuf == NULL) {
    return (*mplexErrHandler) (iSocket);
  }
  retVal = cmNewSearchNExecute(iSocket, sBuf, phtCmds, argDummy);
  free(sBuf);
  return retVal;
}

int
cmNewSearchNExecute(iSocket, sBuf, phtCmds, argDummy)
    int              iSocket;
    char            *sBuf;
    hashTable       *phtCmds;
    char            *argDummy;
{
  struct he      *phe;
  struct cmCommand *pCmd;

  if (sBuf == NULL) {
    fprintf(stderr, "cmNewSearchNExecute()->null input!\n");
    return 0;
  }
  phe = htLookup(phtCmds, sBuf);
  if (phe == NULL) {
    fprintf(stderr, "cmNewSearchNExecute()- Command not found -> %s\n",
        sBuf);
    return (-1);
  }
  pCmd = (struct cmCommand *) phe->data;
#ifdef DEBUG
  fprintf(stderr, "%s\n", pCmd->helpmsg);
#endif              /* DEBUG */
  (*pCmd->function) (iSocket, argDummy);
  return 0;
}


/*------------------------------------------------------------------------
    --
 * cmReceiveString(fd) --
 *-----------------------------------------------------------------------
     --
 */
char           *
cmReceiveString(fd)
    int              fd;
{
  char          *buf;
  int            len, maxlen, c;
  shaString inStr;

  if (mplexTab[fd].fInUse == MPLEX_FREE) {
    fprintf(stderr, "cmReceiveString()-- Bad Channel.\n");
    return NULL;
  }
#ifdef USEXDR
  inStr = NULL;
  if (shaStringIn(fd, &inStr) == -1) {
    perror("shaStringIn()");
    inStr = NULL;
    fprintf(stderr, "CMRS: got (null) on %d\n", fd);
#ifdef DEBUG
#endif              /* DEBUG */
  } else {
#ifdef DEBUG
    len = strlen(inStr);
    fprintf(stderr, "CMRS: (%s) %d on %d\n", inStr, len, fd);
#endif              /* DEBUG */
  }
  return inStr;
#endif              /* USEXDR */

  maxlen = 64;
  len = 0;
  buf = malloc(maxlen);
  do {
    /*
     * Quite inefficient to read byte by byte, but if length is
     * unknown..
     */
#ifdef USE_STREAMS
    c = getc(mplexTab[fd].inStream);
    buf[len] = c;
    if (feof(mplexTab[fd].inStream) && (len == 0))
#else               /* USE_STREAMS */
      if (((c = read(fd, &buf[len], 1)) <= 0) && (len == 0))
#endif              /* USE_STREAMS */
      {
    free(buf);
    return (NULL);
      }
    if (ferror(mplexTab[fd].inStream)) {
      fprintf(stderr, "cmReceiveString()->error on stream of %d\n", fd);
      perror("cmReceiveString()->getc()");
#ifdef WANT
      fprintf(stderr, "mplexTab[%d].inStream = %lx, Base= %lx, Ptr= %lx\n",
          fd,
          mplexTab[fd].inStream,
          mplexTab[fd].inStream->_base,
          mplexTab[fd].inStream->_ptr);
      fprintf(stderr, "mplexTab[%d].inStream Cnt= %d, file= %d, flag= %d\n"
          ,
          fd,
          mplexTab[fd].inStream->_cnt,
          mplexTab[fd].inStream->_file,
          mplexTab[fd].inStream->_flag);
      fprintf(stderr, "mplexTab[%d].inBuf = %lx, (%s), Buf=(%s) len=%d\n",
          fd,
          mplexTab[fd].inBuf,
          mplexTab[fd].inBuf, buf, len);
#endif /*WANT*/
      free(buf);
      return NULL;
      break;
    } else if ((buf[len] == '\0') || (c < 0)) {
      break;
    }
    if (len == maxlen - 1) {
      maxlen *= 2;
      if ((buf = realloc(buf, maxlen)) == NULL) {
    fprintf(stderr, "realloc(): ran out of memory.\n");
    exit(1);
      }
    }
    len++;
  } while (1);          /* TRUE */
  len++;

  if (len < maxlen) {
    if ((buf = realloc(buf, len)) == NULL)
      fprintf(stderr, "warning: realloc failed.\n");
  }
#ifdef DEBUG
  fprintf(stderr, "CMRS: (%s) %d on %d\n", buf, len, fd);
  fprintf(stderr, "mplexTab[%d].inStream = %lx, Base= %lx, Ptr= %lx\n",
      fd, mplexTab[fd].inStream, mplexTab[fd].inStream->_base,
      mplexTab[fd].inStream->_ptr);
#endif              /* DEBUG */
  return (buf);
}

/*------------------------------------------------------------------------
    --
 * cmSendNull(fd) -- send a null character down tube
 *
 *------------------------------------------------------------------------
     --
 */

int
cmSendNull(fd)
    int             fd;
{
  if (mplexTab[fd].fInUse == MPLEX_FREE) {
    fprintf(stderr, "cmSendNULL()-- Bad Channel.\n");
    return -1;
  }
#ifdef USE_STREAMS
  if (fputc(0, mplexTab[fd].outStream) == EOF) {
    return -1;
  }
#else                 /* USE_STREAMS */
  if (write(fd, &c, 1) < 1) {
    return -1;
  }
#endif                /* USE_STREAMS */
  return 0;
}

/*------------------------------------------------------------------------
    --
 * cmSendData(fd, s) -- send a string to a file descriptor, no null at end
 *
 *------------------------------------------------------------------------
     --
 */

int
cmSendData(fd, s)
    int             fd;
    char           *s;
{
  int             n;

  if (mplexTab[fd].fInUse == MPLEX_FREE) {
    fprintf(stderr, "cmSendData()-- Bad Channel.\n");
    return -1;
  }
  n = strlen(s);
#ifdef DEBUG
  fprintf(stderr, "CMSD: (%s) %d on %d\n", s, n, fd);
#endif                /* DEBUG */
#ifdef USEXDR
  if (shaStringOut(fd, &s) == -1) {
    return -1;
  }
  return 0;
#endif                /* USEXDR */
#ifdef USE_STREAMS
  if (fprintf(mplexTab[fd].outStream, "%s", s) == EOF) {
    return -1;
  }
#else                 /* USE_STREAMS */
  if (write(fd, s, n) < n) {
    return -1;
  }
```

```
#endif              /* USE_STREAMS */
  return 0;
}

/*---------------------------------------------------------------------------
    --
 * cmSendString(fd, s) -- send a null-terminated string to a file
     descriptor
 *
 *--------------------------------------------------------------------------
     --
 */

int
cmSendString(fd, s)
    int             fd;
    char           *s;
{
  int             n;
  if (mplexTab[fd].fInUse == MPLEX_FREE) {
    fprintf(stderr, "cmSendString()-- Bad Channel.\n");
    return -1;
  }
  if(s == NULL){
    fprintf(stderr, "cmSendString()-- Sending NULL!!\n");
    s = "";
  }
  n = strlen(s);
#ifdef DEBUG
  fprintf(stderr, "CMSS: (%s) %d on %d\n", s, n, fd);
#endif              /* DEBUG */

#ifdef USEXDR
  if (shaStringOut(fd, &s) == -1) {
    return -1;
  }
  return 0;
#endif              /* USEXDR */
#ifdef USE_STREAMS
  if (fprintf(mplexTab[fd].outStream, "%s", s) == EOF) {
    return -1;
  }
  if (fputc(0, mplexTab[fd].outStream) == EOF) {
    return -1;
  }
#else               /* USE_STREAMS */
  if (write(fd, s, n + 1) < n + 1) {
    return -1;
  }
#endif              /* USE_STREAMS */
  return 0;
}
```

```
/*----------------------------------------------------------------------------
    --
 * cmMultiCast(pfd, nfd, func, arg1, arg2) -- call func on fd list
 *
 *----------------------------------------------------------------------------
     --
 */

int
cmMultiCast(pfd, nfd, func, arg)
    int            *pfd;
    int             nfd;
    int             (*func) ();
    char           *arg;
{
  int             i;
  int             retVal;

  for (i = 0; i < nfd; i++) {
    retVal = (*func) (pfd[i], arg);
  }
  return retVal;
}

cmAckOk(fd)
    int             fd;
{
  return cmSendString(fd, ACK_STRING);
}

cmAckError(fd)
    int             fd;
{
  return cmSendString(fd, ERROR_STRING);
}

int
getServerPort(sService)
    char           *sService;
{
  int iPort;

  if (strcmp(sService, GANITH_NAME) == 0) {
    iPort = GANITH_PORT;
  } else if (strcmp(sService, SHILP_NAME) == 0) {
    iPort = SHILP_PORT;
  } else if (strcmp(sService, VAIDAK_NAME) == 0) {
    iPort = VAIDAK_PORT;
  } else if (strcmp(sService, SHASTRA_NAME) == 0) {
    iPort = SHASTRA_PORT;
  } else if (strcmp(sService, SCULPT_NAME) == 0) {
    iPort = SCULPT_PORT;
  } else if (strcmp(sService, BHAUTIK_NAME) == 0) {
      iPort = BHAUTIK_PORT;
    } else if (strcmp(sService, SPLINEX_NAME) == 0) {
      iPort = SPLINEX_PORT;
    } else if (strcmp(sService, GATI_NAME) == 0) {
      iPort = GATI_PORT;
    } else if (strcmp(sService, VOLREND_NAME) == 0) {
      iPort = VOLREND_PORT;
    } else if (strcmp(sService, SHLISP_NAME) == 0) {
      iPort = SHLISP_PORT;
    } else {
      iPort = 0;
  #ifdef DEBUG
      fprintf(stderr, "getServerPort()->Unknown Service %s\n", sService);
  #endif /* DEBUG */
    }
  #ifdef DEBUG
    fprintf(stderr, "getServerPort()->Using iPort %d for %s\n",
        iPort, sService);
  #endif /* DEBUG */
    return iPort;
  }

  static void ModelHandler(fd)
  int fd;
  {
   char *arg;
   int status = 0;

   arg = cmReceiveString(fd);
   /* ...handler code ... */
   status = 1;
   if (status){
      cmAckOk(fd);
   }else{
    cmAckError(fd);
    free(arg);
   }
  }

  /*
   * readString(iSocket) - read string from interface
   */
  char           *
  readString(iSocket)
      int              iSocket;
  {
    int              fBlank;
    int              i;
    int              n;
    char            *sbIO;

    fBlank = 1;
    while (fBlank) {
```

```
    sbIO = cmReceiveString(iSocket);
    n = strlen(sbIO);
    for (i = 0; i < n; i++) {
      if (!isspace(sbIO[i])) {
    fBlank = 0;
    break;
      }
    }
    if (fBlank) {
      free(sbIO);
    }
  }
#if RS_DEBUG
  fprintf(stderr, "readString: %s", sbIO);
#endif
  return (sbIO);
}

/*
 * readStrings(iSocket) - read n strings and return ptr to char ** array
 * expects #, string ...
 */
char          **
readStrings(iSocket)
    int             iSocket;
{
  char          **names;
  int             number, i;
  char           *sbIn;
  int             len;

  sscanf((sbIn = readString(iSocket)), "%d", &number);
  free(sbIn);

  if (number <= 0) {
    return NULL;
  }
  names = (char **) malloc((1 + number) * sizeof(char *));

  for (i = 0; i < number; i++) {
    names[i] = readString(iSocket);
    len = strlen(names[i]);
    if (names[i][len - 1] == '\n') {
      names[i][len - 1] = '\0';
    }
  }
  names[number] = NULL;

  return (names);
}

/*
 */
```

```
int
cmFlush(fd)
    int             fd;
{

  if (mplexTab[fd].fInUse == MPLEX_FREE) {
    return -1;
  }
#define WANTnn
#ifdef WANT
  int             base, ptr, cnt, diff;
  unsigned int    posn;
  posn = xdr_getpos(mplexXDRSEnc(fd));
  cnt = mplexTab[fd].outStream->_cnt;
  base = (int) mplexTab[fd].outStream->_base;
  ptr = (int) mplexTab[fd].outStream->_ptr;
  diff = ptr - base;
  if (diff == 0) {
    fprintf(stderr, "younds! diff is 0\n");
  }
  fprintf(stderr, "(Bef)outPos= %u, Cnt= %d, Base= %lx, Ptr= %lx, Diff= %d\
      n",
      posn, cnt, base, ptr, diff);

  posn = xdr_getpos(mplexXDRSDec(fd));
  cnt = mplexTab[fd].inStream->_cnt;
  base = (int) mplexTab[fd].inStream->_base;
  ptr = (int) mplexTab[fd].inStream->_ptr;
  diff = ptr - base;
  fprintf(stderr, "      InPos= %u, Cnt= %d, Base= %lx, Ptr= %lx, Diff= %d\
      n",
      posn, cnt, base, ptr, diff);
#endif              /* WANT */

#ifdef USE_STREAMS
  /*
    fprintf(stderr, "mplexTab[%d].outStream->_cnt = %d, diff = %d\n",fd,
    mplexTab[fd].inStream->_cnt,
    mplexTab[fd].inStream->_ptr-mplexTab[fd].inStream->_base);
    */
  if (fflush(mplexTab[fd].outStream) == EOF) {
    perror("fflush()");
    return -1;
  }
#endif              /* USE_STREAMS */

#ifdef WANT
  posn = xdr_getpos(mplexXDRSEnc(fd));
  cnt = mplexTab[fd].outStream->_cnt;
  base = (int) mplexTab[fd].outStream->_base;
  ptr = (int) mplexTab[fd].outStream->_ptr;
  diff = ptr - base;
  fprintf(stderr, "(Aft)outPos= %u, Cnt= %d, Base= %lx, Ptr= %lx, Diff= %d\
      n",
      posn, cnt, base, ptr, diff);

#endif              /* WANT */

}


cmMain(argc, argv)
    int             argc;
    char          **argv;
{
  int             iSocket;
  int             iSockNew;
  struct sockaddr_in saInNew;
  int             iLength, iOption;

  cmOpenServerSocket("shilp", 0, NULL, &iSocket, NULL);

#ifdef DEBUG
  fprintf(stderr, "Tcp Socket has port #%d\n", ntohs(saInServer.sin_port));
  fprintf(stderr, "Got socket descr %d for connect\n", iSocket);
#endif              /* DEBUG */
  if (listen(iSocket, 5) != 0) {
    perror("listen()");
    return -1;
  }
  iOption = 1;
  if(setsockopt(iSocket, SOL_SOCKET, SO_REUSEADDR, &iOption,
        sizeof(iOption)) != 0){
    perror("setsockopt() SOL_SOCKET, SO_REUSEADDR");
    return -1;
  }
  /* allow socket to be reused locally, foreign diff */

  if ((iSockNew = accept(iSocket, &saInNew, &iLength)) < 0) {
    perror("accept()");
    return -1;
  }
#ifdef DEBUG
  fprintf(stderr, "Got socket descriptor %d for client connection\n",
      iSockNew);
#endif              /* DEBUG */
  close(iSocket);
  fprintf(stderr, "%d, %s, %d\n", argc, argv[0], iSockNew);
  return 0;
}
```